


CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR

Editorial Delta, S.A.

# MI COMPUTER CURSO PRACTICO
## DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR


Publicado por Editorial Delta, S.A., Barcelona

Volumen IX-Fascículo 99

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 -


MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.



ISBN: 84-85822-83-8 (fascículo) 84-7598-181-X (tomo 9)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 188512
Impreso en España-Printed in Spain-Diciembre 1985


Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.


Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.


Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

**Servicio de suscripciones y atrasados** (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

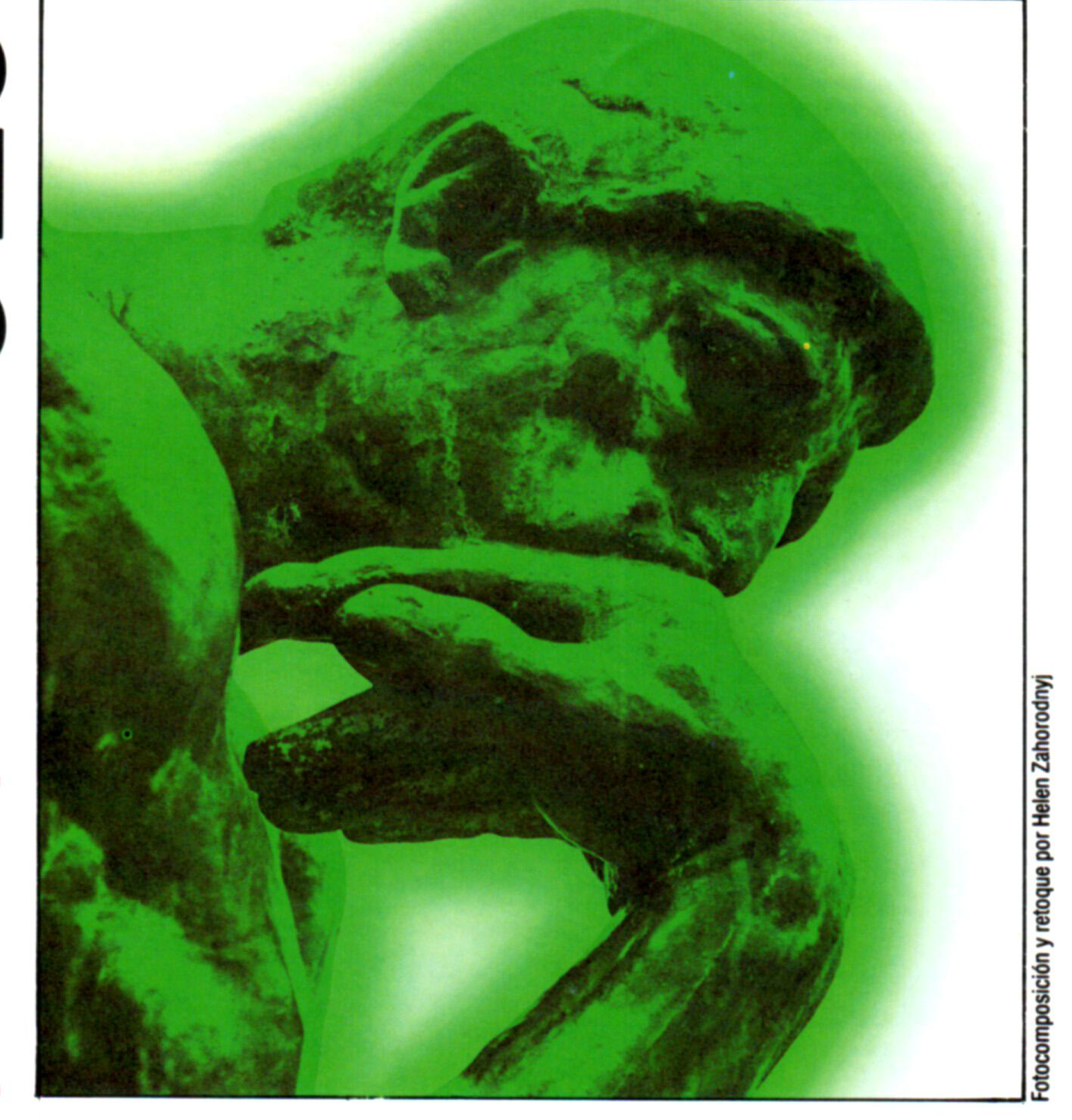
Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

**No se efectúan envíos contra reembolso.**





# Mirando hacia el futuro



Fotocomposición y retoque por Helen Zahorodnyj

**¿Es el hombre sólo una fase evolutiva en un proceso que conduce a un mundo dominado por máquinas inteligentes? Consideremos los fundamentos de esta inquietante hipótesis**

Si consideramos los desarrollos de la informática como si se tratara de la ascensión por una escalera, podemos distinguir, en términos generales, cuatro peldaños de creciente complejidad. En el peldaño inferior tenemos las tareas rutinarias de contabilidad que los ordenadores realizan tan bien. Un tramo más hacia arriba, las cosas se ponen un poco más interesantes. Aquí se hallan los programas que ayudan a las personas a tomar decisiones inteligentes: los sistemas de previsión financiera y la distribución electrónica de información impresa, p. ej. Estas herramientas han de ser flexibles, ya que las demandas del usuario son imprevisibles.

En el tercer nivel encontramos la aplicación de pericia derivada de los seres humanos. Es aquí donde se concentra el grueso de las aplicaciones de la AI, tanto ahora como en un futuro inmediato. Un ejemplo de este tipo de software es el sistema experto PROSPECTOR, que codificó las conclusiones de las investigaciones de varios geólogos. Al aplicar estas reglas en el campo, sus autores se vieron recompensados con el descubrimiento de un enorme depósito desconocido de molibdeno en el estado de Washington, al noroeste de Estados Unidos. Poco después se descubrió otro en Canadá.

Las realizaciones del PROSPECTOR han pasado a formar parte de la mitología de la AI, y, por cierto, son impresionantes. Sin embargo, no nos llevan hasta el cuarto peldaño de la escalera: la producción de máquinas con inteligencia creadora. Para comprender en qué consiste esto, es necesario delimitar claramente productividad y creatividad.

La productividad se basa en seguir las reglas, y los ordenadores son excelentes seguidores de éstas. Pueden ser más productivos que las personas; pero dotarlos de creatividad ha resultado ser increíblemente difícil. Para ser auténticamente creadores, los programas para ordenador habrán de producir sus propias reglas. Ciertos investigadores de las fronteras de la AI están, sin embargo, tratando de hacer que los ordenadores hagan exactamente eso.

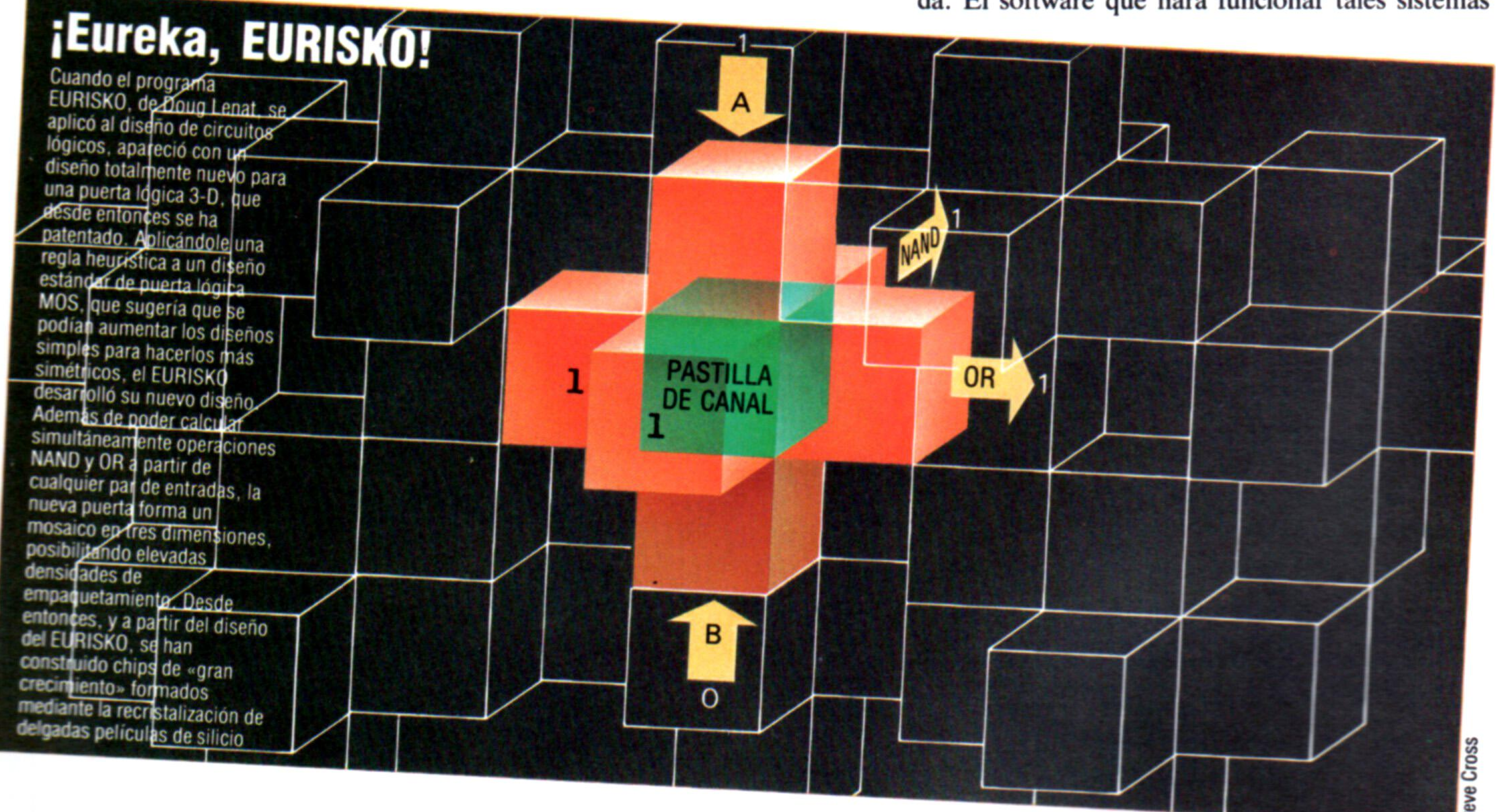
Un programa que ha dado un paso adelante hacia el cuarto peldaño de la escalera es EURISKO, desarrollado por Doug Lenat en la Universidad de Stanford. EURISKO es un programa de descubrimiento que ya hemos mencionado con anterioridad en esta serie. Se ha aplicado a varios campos, desde un juego de guerra naval hasta el diseño de circuitos VLSI (integración a muy gran escala). EURISKO parte de un conjunto de reglas y conceptos heurísticos y los aplica a los dominios escogidos. Hasta aquí todo es normal: su novedad radica en el hecho de que puede modificar su propia heurística. Ello le confiere al sistema un enorme poder, dado que puede adaptar sus reglas generales y especializarlas para afrontar situaciones nuevas.

EURISKO ha hecho un descubrimiento por el cual posteriormente se otorgó una patente, de modo que no se lo podría considerar como superficial. Se trata de un nuevo diseño para un circuito lógico 3-D (tridimensional, formado por una puerta NAND/OR) inédito en Silicon Valley —California— (y en todas partes). Y fue generado por la aplicación de una única regla.

Cuando se le aplicó el EURISKO al diseño VLSI, ya contaba con una regla heurística, obtenida a partir de tareas previas, que afirmaba: «si un concepto es interesante, intenta hacerlo más simé-

**Realidad más allá de la imaginación**
El progreso en el desarrollo de la biotecnología podría conducir a la inquietante perspectiva de los híbridos hombre-máquina, mezclando los procesos intuitivos del cerebro humano con el potencial lógico y matemático de los microprocesadores. En el caso de que esto ocurriera, podríamos ver al proceso de la evolución humana arrancado de las manos de la naturaleza y puesto bajo el control de hombres-máquina capaces de una autorreproducción selectiva

trico». Aplicada a un dispositivo 2-D (bidimensional), condujo a una versión 3-D más simétrica de ese dispositivo, el cual se ha fabricado recientemente con gran éxito. Un dispositivo 3-D, una vez sorteados los problemas de fabricación, se puede empaquetar más densamente, ofreciendo, por lo tanto, una mayor capacidad.




## ¡Eureka, EURISKO!

Cuando el programa EURISKO, de Doug Lenat, se aplicó al diseño de circuitos lógicos, apareció con un diseño totalmente nuevo para una puerta lógica 3-D, que desde entonces se ha patentado. Aplicándole una regla heurística a un diseño estándar de puerta lógica MOS, que sugería que se podían aumentar los diseños simples para hacerlos más simétricos, el EURISKO desarrolló su nuevo diseño. Además de poder calcular simultáneamente operaciones NAND y OR a partir de cualquier par de entradas, la nueva puerta forma un mosaico en tres dimensiones, posibilitando elevadas densidades de empaquetamiento. Desde entonces, y a partir del diseño del EURISKO, se han construido chips de «gran crecimiento» formados mediante la recristalización de delgadas películas de silicio

Steve Cross

La creatividad se considera la cumbre de la inteligencia humana, envuelta en un velo de misteriosas seudoexplicaciones que entrañan intuición y penetración; pero EURISKO habrá de concedernos una pausa para reflexionar. He aquí un ejemplo de un auténtico descubrimiento obtenido a partir de la aplicación de una sola regla. El ordenador creativo está más cerca de lo que se cree comúnmente.

A lo largo de esta serie, hasta ahora nos hemos venido concentrando en el rostro aceptable de la AI. La hemos visto como la vanguardia de la ciencia informática: una fuente fértil de ideas nuevas y de recursos inteligentes. Cuando estas ideas tienen éxito, se traspasan a otras áreas de la informática. Esto ha sucedido en el pasado con el procesamiento de listas y la informática conversacional, y actualmente está sucediendo con los sistemas basados en el conocimiento. La AI «exporta sus éxitos».

Las perspectivas a medio plazo para este tipo de trabajo son buenas. Podemos esperar progresar en un frente amplio. Puede haber reveses, y puede ser que algunas de las predicciones más atrevidas no se materialicen, pero a finales de este siglo habremos sido testigos de sustanciales avances en los campos de la visión por ordenador, la traducción automática, el aprendizaje de la máquina y los sistemas basados en el conocimiento. Incluso el espinoso problema de la comprensión del habla continua estará próximo a su solución.

La iniciativa japonesa de la quinta generación le ha proporcionado un extraordinario impulso a esta cara de la AI. A raíz de sus ambiciosos planes, los japoneses han redefinido sutilmente las reglas del juego para la industria del ordenador. Para 1990 esperan producir procesadores de información del conocimiento basados en arquitecturas informáticas radicalmente nuevas, y paralelas en gran medida. El software que hará funcionar tales sistemas está firmemente enraizado en la investigación sobre AI. La confianza de los japoneses en las técnicas de AI ha hecho que el resto del mundo se vuelque a ella. Se ha convencido a los gobiernos para que aprueben planes de inversión importantes, a fin de no quedarse rezagados en este terreno.

Existe, sin embargo, lo que para algunas personas es la cara oscura de la AI. Independientemente de lo notables que sean los adelantos en el campo de la inteligencia artificial, quizá no nos agraden algunos de los usos a los que se destinen, en especial si consideramos hasta qué punto depende de los fondos militares la investigación en AI. Más de la mitad de las aplicaciones a corto plazo son de carácter bélico. Éstas incluyen:

- Submarinos inteligentes
- Municiones «sagaces»
- Misiles crucero
- Carros de combate autodirigidos
- Sistemas de sonar basados en el conocimiento
- Torpedos autodireccionales
- Sistemas de imágenes por radar

y muchas aplicaciones más sobre las que la mayoría de nosotros no habrá siquiera oído hablar.

Consideremos por un momento qué es lo que hace que una granada de artillería sea «sagaz». No se limita a caer desde el aire y abrir un agujero en el suelo: selecciona su objetivo. Al acercarse al fin de su vuelo busca objetivos con aspecto de carro de combate y ajusta su trayectoria para asegurarse de que caiga sobre uno de ellos. Dispare en la dirección general adecuada y, con toda seguridad, hará blanco. Así es una de las aplicaciones inmediatas de

la AI: mejores formas de destruir carros de combate.

Aún más alarmantes son las tramas de anticipación que han urdido algunos científicos a partir de éxitos notabilísimos en la búsqueda de *máquinas ultrainteligentes* (UIM: *ultra-intelligent machines*), tal como se ha dado en llamar a los artefactos dotados de inteligencia «suprahumana». Estos hombres de ciencia prevén un futuro dominado por máquinas UIM. Sin «pensar» realmente, estas máquinas serán capaces de hacer casi todo lo que exige un pensamiento razonado y mejor de lo que pueden hacerlo las personas. Nos aventajarán muchísimo en matemáticas y ciencias naturales. En la industria serán unos administradores tan efectivos, que el manejo de economías completas estará bajo su control. En resumen, seremos inferiores intelectualmente. Esta predicción considera una falacia la sugerencia de que siempre podremos «tirar del enchufe» en el caso de que la inteligencia de las máquinas llegara a tal punto que éstas se constituyeran en entes incontrolables.

Durante mucho tiempo, uno de los conceptos recurrentes en las obras de ciencia-ficción ha sido la de los híbridos hombre-máquina; pero sólo recientemente esta idea ha traspasado la frontera de lo fantástico entrando en el área de la conjetura científica a largo plazo. La influencia de dos áreas de la investigación científica aparentemente incompatibles (la ingeniería genética y la ingeniería del conocimiento) es en gran parte responsable de ello.

La ingeniería genética trata de la manipulación del código genético en la materia viva para «programar» mejoras en el campo de la herencia en las generaciones sucesivas. Los ingenieros genéticos ya han perfeccionado técnicas que permiten trasplantar características deseadas en microorganismos ya existentes con el objeto de producir drogas que con anterioridad eran prohibitivas por su precio o bien imposibles de producir a gran escala. Existe la propuesta de que, en un futuro cercano, los métodos de la ingeniería genética serán lo suficientemente sofisticados como para producir los llamados *biochips*. Mediante la programación de los enzimas que dividen y reconstituyen las moléculas se podrá *hacer crecer* circuitos lógicos. La mejora en la densidad de empaquetamiento, obtenida haciendo crecer circuitos a partir de moléculas de proteínas en vez de crearlos utilizando los métodos actuales, tendría como consecuencia una drástica reducción en cuanto a tamaño. La idea de que la materia viva transporte mensajes eléctricos codificados no es nueva: es el mecanismo básico en función del cual trabaja el sistema nervioso humano.

En la actualidad, los ingenieros genéticos alteran la estructura genética del trigo para hacerlo más resistente al ataque de los hongos, pero estas técnicas se podrían utilizar fácilmente para erradicar enfermedades genéticas, como el síndrome de Down, en los seres humanos. Una vez dados los primeros pasos en la manipulación de los genes humanos, a la larga podrían efectuarse otros experimentos; por ejemplo, los científicos podrían tratar de trasplantar en el cerebro humano una bioROM que contenga una base de datos de información. Aunque esto pueda parecer rebuscado, los científicos ya han obtenido cierto éxito implantando aparatos eléctricos en el cerebro. A una estudiante sorda de Oxford se le ha enviado a las áreas auditivas del cerebro la salida eléctrica de un micrófono. Tras un corto período de entrenamiento, pudo descifrar el significado de estas señales y "oír" sonidos.

Algunas personas plantean que en el futuro lejano, nuestros descendientes evolutivos puedan ser híbridos hombre-máquina sumamente avanzados. Puesto que el proceso evolutivo parece hacer uso del material que tenga a mano, sea cual fuere (las aletas se convirtieron en patas; los pulmones que desarrollaron los peces para la flotación se utilizan para respirar aire), parece razonable suponer que estos organismos híbridos pudieran tener en su corazón un cerebro humano rodeado por muchas capas de tecnología avanzada. En realidad, el cerebro humano en su forma actual se compone de varias capas que se desarrollaron durante nuestra evolución.

**De máquina a superhombre**
HAL, el ordenador que coprotagonizó la película *2001: una odisea del espacio*, y que vemos aquí en un fotograma de *2010: el año en que hicimos contacto*, secuela de la anterior, comienza a vivir equipado con unas amplias facilidades para síntesis de voz y un complejo juego de reglas en función de los cuales juzga el éxito de una misión. Tales configuraciones se están implementando cada vez más en el hardware y el software de hoy en día. En la novela de igual título en que se inspiró la segunda de las películas citadas, el autor, Arthur Clarke, llevó el concepto de inteligencia de la máquina varios pasos más hacia adelante, al hacer que HAL trascienda su condición mecánica para convertirse en una figura suprahumana, cuestionando, en consecuencia, el supuesto de que las máquinas no pueden evolucionar independientemente de sus creadores

Cortesía de U.I.P.

**Establecimiento de la pendiente para la patilla BUSY**
Las fases I y II del proceso de conversión tienen una duración fija, pero el tiempo que lleva efectuar la tercera fase es proporcional a la magnitud de la tensión de entrada original, $V_{ent}$. Durante esta fase, la tensión de entrada previamente integrada se reduce uniformemente a cero: cuanto mayor sea la tensión inicial, más tiempo se empleará en este proceso. Este hecho nos permite conectar nuestro DVM en interface con un ordenador. La patilla BUSY del chip convertidor 7135 se hace alta (*high*) al comienzo de la fase de integración de señal y permanece así hasta un impulso de reloj después de que la salida del integrador pasa por cero. Si con una AND se relacionan las señales BUSY y CLOCK mediante una puerta lógica, la salida tomará la forma de impulsos de reloj válidos, es decir, impulsos de reloj entre el comienzo de la segunda fase y el final de la tercera. Estos impulsos se pueden transmitir a un ordenador a través de un enlace en serie. Tras restar 10 001 impulsos de reloj para tener en cuenta la fase "integración de la señal" y el impulso nulo al final de la fase "integración de la referencia", la cantidad de impulsos restante será directamente proporcional a la tensión de entrada original. Por lo tanto, tras el calibrado, 20 000 impulsos corresponderán a 2 V, y así sucesivamente

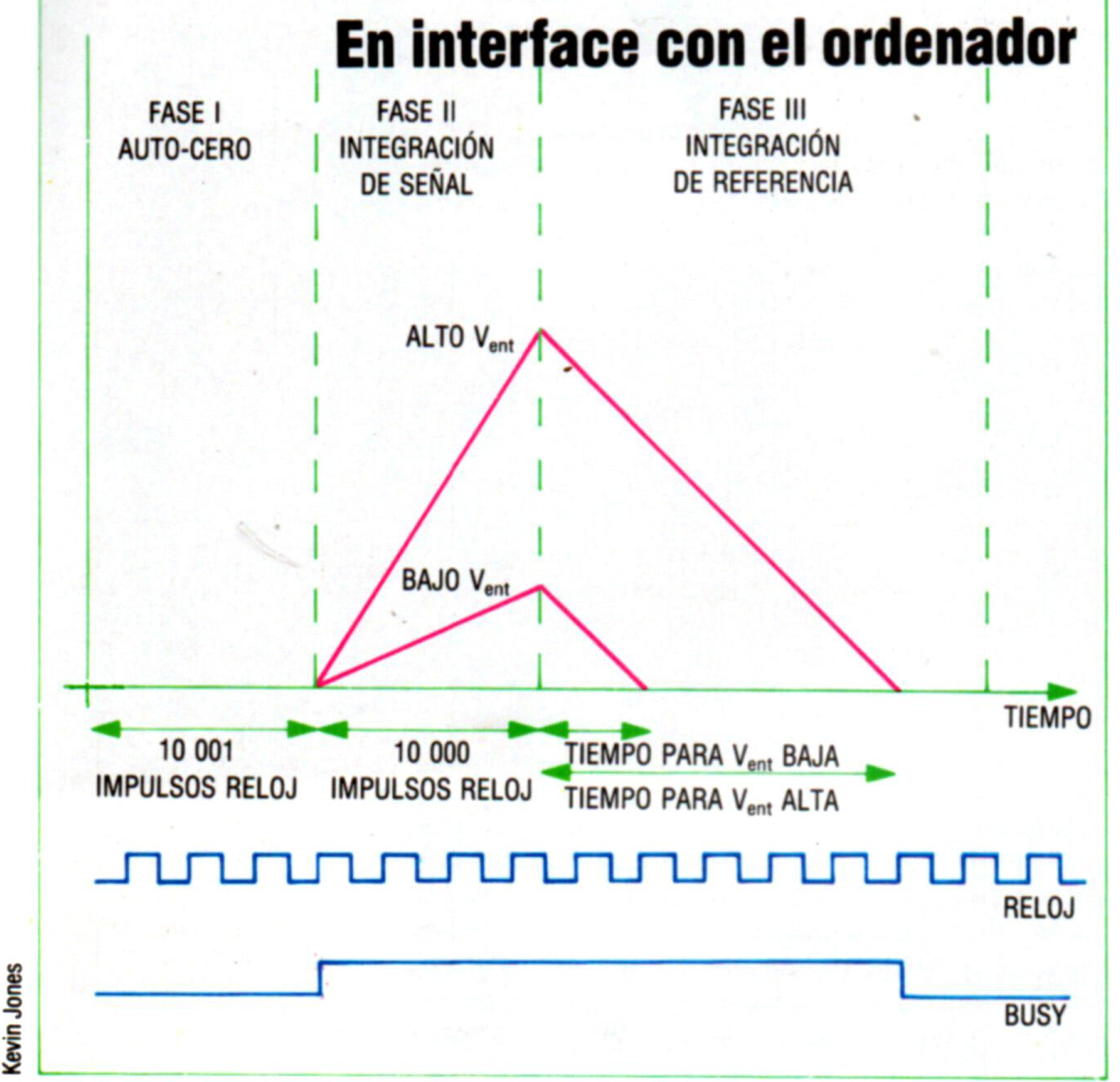
# Conversión integradora

## Examinemos de cerca el chip integrador 7135, localizado en el centro del tester digital

La mayoría de los convertidores A/D, ya sea los construidos a partir de componentes discretos (un hecho raro en estos días) o bien los integrados en un solo chip, por lo general utilizan uno de dos métodos. Se dice que los dispositivos que incorporan estos métodos son *convertidores de aproximación sucesiva* y *convertidores integradores*. El chip 7135 es un convertidor de tipo integrador, pero antes de examinar su funcionamiento de forma detallada, veamos brevemente cómo trabajan los dos tipos.

Los convertidores A/D de aproximación sucesiva se basan en un convertidor D/A utilizado en un bucle de realimentación junto con un comparador. La salida del convertidor D/A se produce de a un bit por vez, comenzando por el MSB (*most significant bit*: bit más significativo) y avanzando hacia el LSB (*least significant bit*: bit menos significativo). Todos los bits de la palabra del convertidor D/A se fijan inicialmente en uno. A medida que se van efectuando las comparaciones, el bit que se esté considerando se deja en uno si la salida del convertidor D/A es menor que la tensión de entrada, y se establece en cero si la salida D/A es mayor que la entrada. Tras cada comparación, se compara el siguiente bit menos significativo. Tras comprobar todos los bits disponibles (por lo general 8, 12 o 16), los bits que quedaron en estado «uno» permiten que circule una corriente desde el convertidor D/A, que corresponde a la entrada analógica.

Una conversión de ocho bits sólo lleva ocho comparaciones, la conversión de 16 bits sólo lleva 16, y así sucesivamente. Esto hace que la técnica de aproximación sucesiva sea sumamente rápida (100 000 conversiones por segundo o más) y, por lo tanto, ideal para conversiones de audio, video, radar, etc., en las que en un lapso muy breve se han de convertir a forma digital muchísimos datos analógicos. Esta técnica, no obstante, plantea numerosos inconvenientes, dos de los cuales son que los convertidores de aproximación sucesiva son caros y requieren un complicado sistema de circuitos.




## Convertidores integradores

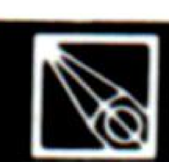
Cuando no son esenciales las conversiones a gran velocidad (los voltímetros digitales constituyen un buen ejemplo de ello), el diseño elegido por lo general es el convertidor integrador. Cuando no se pretende digitalizar formas de onda CA de elevada frecuencia, son muy adecuadas velocidades de muestreo de unas pocas lecturas por segundo.

La salida de un convertidor A/D integrador representa el valor promedio de una tensión de entrada analógica a través de un período de tiempo fijo. A diferencia de los convertidores A/D de aproximación sucesiva, que deben «muestrear y retener» la señal de entrada antes de que tenga lugar la comparación y conversión, los de tipo integrador digitalizan la entrada utilizando el tiempo. Para temporizar la conversión se pueden usar señales de reloj.

Entre las ventajas de la técnica del convertidor integrador, que es la que utilizaremos para nuestro diseño, se incluyen una gran precisión, componentes no críticos (aparte de los componentes de la tensión de referencia), excelente rechazo del ruido, ausencia de la necesidad del difícil sistema de circuitos para «muestreo y retención», y componentes de costo comparativamente reducido.

En un circuito convertidor A/D de tipo integrador típico, la conversión se produce en tres fases. Éstas se conocen como la fase de *auto-cero*, la fase de *integración de la señal* y la fase de *integración de la referencia*. Éstas son, por consiguiente, relativamente inmunes a los efectos de fluctuaciones de elevada frecuencia en el "ruido" de entrada (al contrario que los convertidores de aproximación sucesiva para "muestrear y retener").

Todo cuanto se requiere para un convertidor A/D integrador, aparte de una señal de reloj estable, es una tensión de referencia de precisión. Ésta se obtiene, como veremos más adelante, utilizando diodos de referencia de intervalo de banda, que se



consiguen con toda facilidad. Estos diodos limitan la tensión máxima que se pueda aplicar.

La mayoría de los convertidores A/D integradores, incluyendo al 7135, utilizan la técnica denominada de *atenuación doble*. Las tres fases operan así:

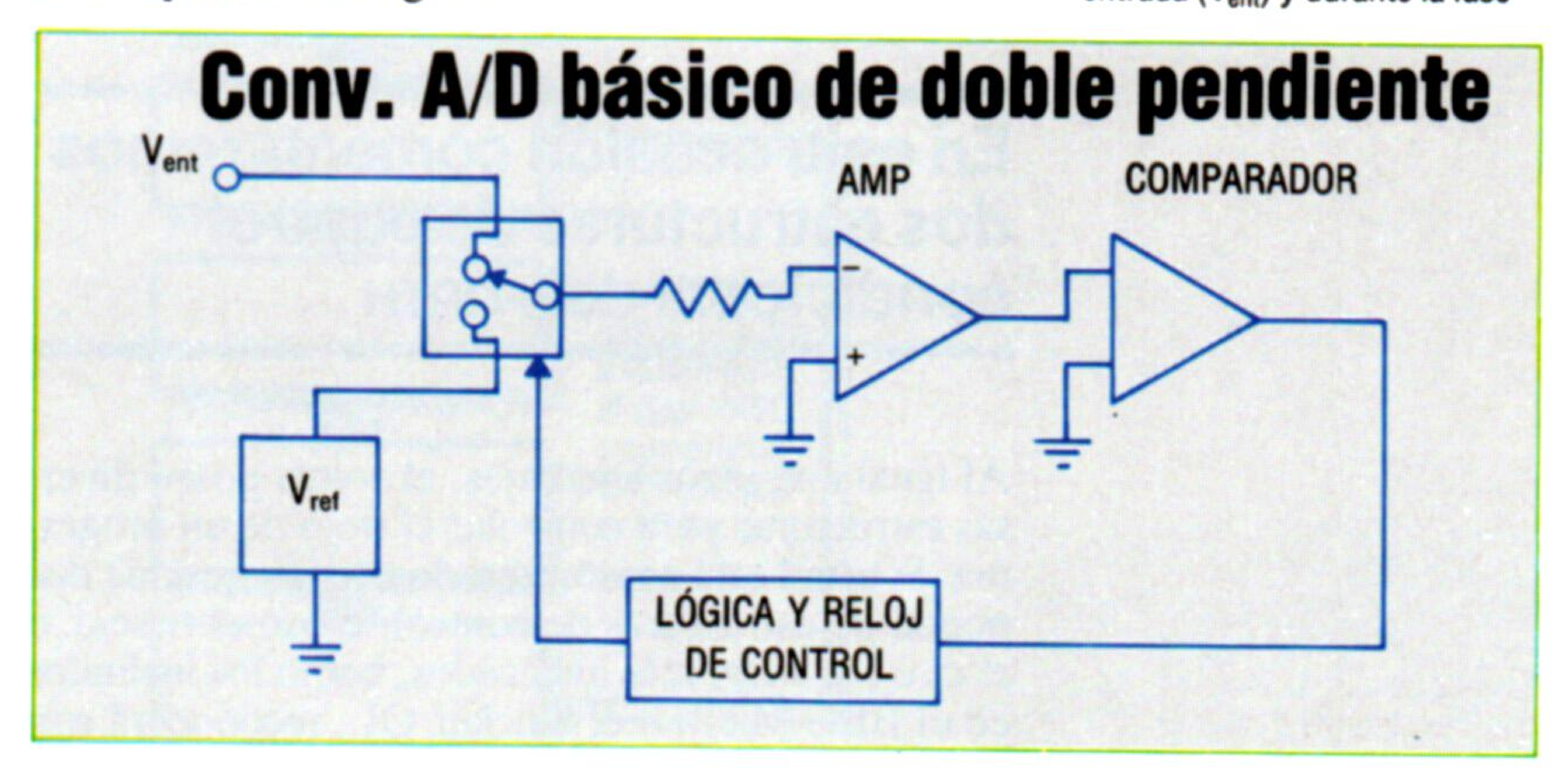
*Fase 1: auto-cero*: Se anulan los errores de los componentes analógicos derivando la entrada a tierra y almacenando la información de error en un condensador de auto-cero.
*Fase 2: integración de la señal*: Se integra la señal de entrada durante cierta cantidad de impulsos del reloj; para un convertidor de 4,5 dígitos, lo típico son 10 000 impulsos de reloj. Una vez terminado el período de integración, la tensión obtenida es directamente proporcional a la señal de entrada.
*Fase 3: integración de la referencia*: Al comienzo de la fase, la entrada al integrador se conmuta desde la tensión de entrada a medir, hasta la tensión de referencia. La cantidad de impulsos de reloj contados desde el principio de la fase integración de referencia hasta el momento en el que la salida del integrador pasa por cero, representa la magnitud de la señal de entrada.

Los convertidores A/D de atenuación doble son intrínsecamente exactos, ya que todo el proceso depende exclusivamente de la absoluta precisión de la tensión de referencia (por ello se debe escoger con cuidado el diodo zener de referencia) y la calidad de los impulsos del reloj. No es necesario que el reloj tenga una frecuencia determinada, ni que cada impulso tenga exactamente la misma duración que los otros. Una de las razones por las que hemos optado por usar el chip temporizador 555 para el reloj es, precisamente, que se trata de un dispositivo básicamente estable y exacto.

La estabilidad de los otros componentes, como el condensador de integración, no tiene especial relevancia a condición de que su valor no cambie durante ninguno de los ciclos de conversión (los convertidores de *aproximación sucesiva*, por el contrario, basan su precisión en la correspondencia exacta de las resistencias de una "escalera" de resistencias). Puesto que es fácil mantener estabilizada la exactitud del reloj en menos de una parte por millón, los convertidores A/D de pendiente doble ofrecen muchísimas ventajas sobre los convertidores de aproximación sucesiva, siempre que no se requiera una gran velocidad de conversión.

## El chip 7135

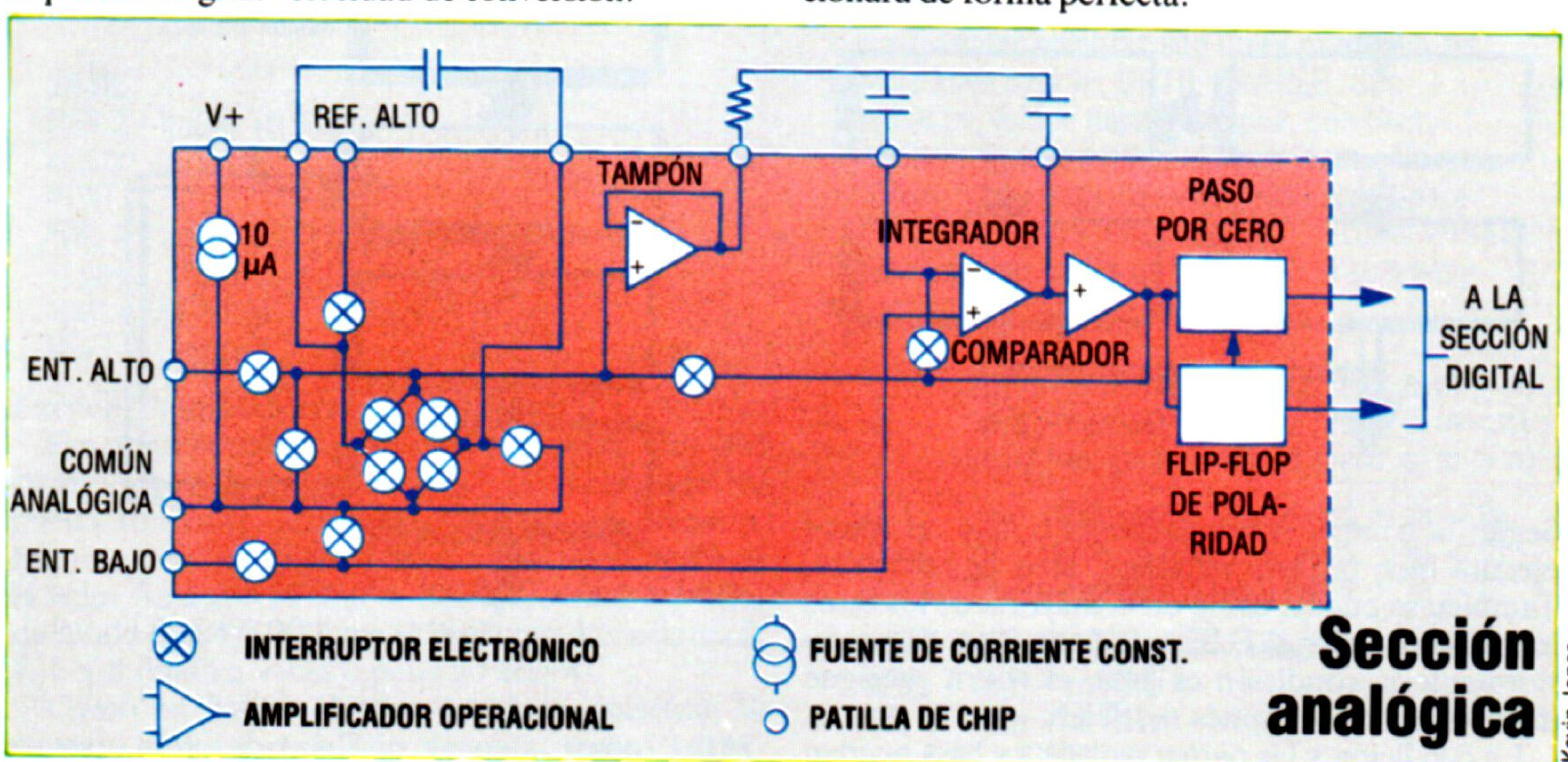
En la ilustración vemos la sección analógica del 7135. La mayoría de los símbolos probablemente le resultarán familiares, exceptuando, quizá, el doble círculo y los círculos con cruces. Éstos representan, respectivamente, fuentes de corriente constante e interruptores analógicos.




**Alrededor del circuito**
Un convertidor A/D integrador de atenuación doble lleva a cabo su tarea en tres fases diferentes: auto-cero, integración de la señal e integración de la referencia. Durante la fase de integración de la señal, el circuito conmuta la tensión de entrada ($V_{ent}$) y durante la fase de integración de la referencia se conmuta la tensión de referencia utilizada para calibrar el convertidor ($V_{ref}$). Una fuente de reloj externa coordina toda la operación

Durante las tres fases de la acción del convertidor (auto-cero, integración de señal e integración de referencia) es importante observar que la cantidad de impulsos de reloj en el tiempo es fija para las dos primeras, y variable para la última. El autocero emplea 10 001 impulsos de reloj, la integración de la señal, 10 000 impulsos, y la integración de la referencia, tantos impulsos como sea necesario para que la salida del integrador pase de cero (hasta un máximo de 20 001). Esto permite aplicar una técnica muy sencilla para que el ordenador lea la salida A/D, utilizando una interface en serie.

La frecuencia del reloj, dentro de unos límites amplios, no tiene importancia. Bastarán frecuencias de reloj tan bajas como de 5 KHz (dando un ciclo de medición de alrededor de 10 segundos), o bien algunas tan altas como de 1 MHz. No obstante, las velocidades de reloj muy bajas producen errores debido a las fugas de los condensadores de referencia, y las velocidades de reloj muy altas requieren una ingeniosa compensación del condensador integrador usando resistencias emparejadas cuidadosamente. Basta decir que cualquier frecuencia de reloj de entre 100 KHz y 160 KHz funcionará de forma perfecta.




**Paso analógico**
El diagrama ilustra el trazado de los principales componentes analógicos existentes en el chip convertidor A/D 7135. Los componentes incluidos dentro de la superficie sombreada se hallan realmente "dentro" del chip. Todos los otros componentes son externos. La sección analógica del 7135 es esencialmente un circuito convertidor A/D integrador de doble pendiente, que le proporciona a la sección digital del chip señales de polaridad y paso por cero. En conjunción con las señales externas de reloj, éstas son suficientes para proporcionar a la sección digital la información que necesita para generar la salida BCD digital del chip

# Con dos condiciones

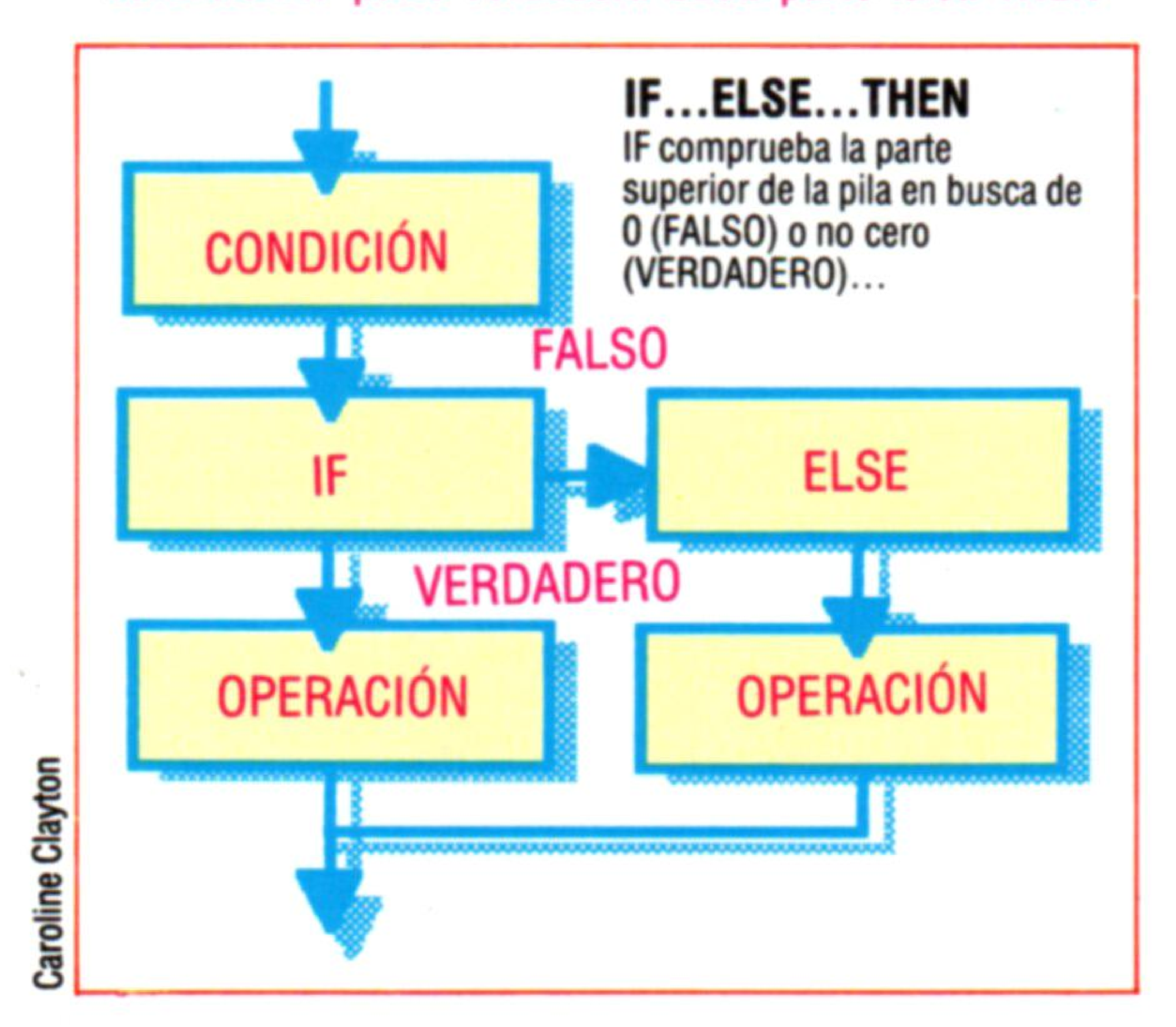
## En esta ocasión comentaremos dos estructuras de control condicional del FORTH

Al igual que otros lenguajes, el FORTH posee diversas estructuras para controlar el flujo de un programa. Si usted está acostumbrado a lenguajes que disponen de estructuras de control, como el PASCAL o el C, o los BASIC más avanzados, como los incluidos en el BBC Micro o el Sinclair QL, reconocerá con facilidad lo que las mismas hacen. Sólo recuerde, sin embargo, que debido a la forma en que el FORTH utiliza una pila, a menudo estas estructuras parecen estar escritas al revés.

Otro punto a tener presente es que estas estructuras sólo se pueden emplear dentro de definiciones de dos puntos. Esto se debe a que el FORTH necesita compilar las instrucciones de bifurcación en saltos condicionados y absolutos. Este proceso requiere una cierta cantidad de tiempo, que no estaría disponible si hubiese que compilar las instrucciones durante la ejecución del programa. Insertando las estructuras dentro de definiciones de dos puntos, usted le brinda al FORTH la posibilidad de calcular exactamente lo que está sucediendo mientras la definición está todavía incorporada en el diccionario.

El FORTH estándar dispone de lo siguiente:

- condición IF parte verdadera ELSE parte falsa THEN




Caroline Clayton

Según la condición sea verdadera o falsa, el FORTH ejecuta bien la parte verdadera, bien la parte falsa. También se puede, como en la mayoría de los otros lenguajes, omitir el ELSE y la parte falsa. Si posteriormente la condición es falsa, el FORTH prosigue inmediatamente después del THEN.

La condición y las partes verdadera y falsa pueden ser cualquier secuencia de palabras del FORTH, posiblemente con más IF...THEN...ELSEs. La condición ha de estar en la pila, y luego el IF la sacará. He aquí un ejemplo, que toma un número de la pila e imprime si es impar o par:

```
:PARIDAD      ( n--)
              (imprime n "es par" o "es impar")
  DUP.        (imprime el propio n)
  ."es"
  2 MOD 0 = IF
    ."par"
  ELSE
    ."impar"
  THEN
;
```

MOD da el resto cuando se divide el primer número por el segundo.

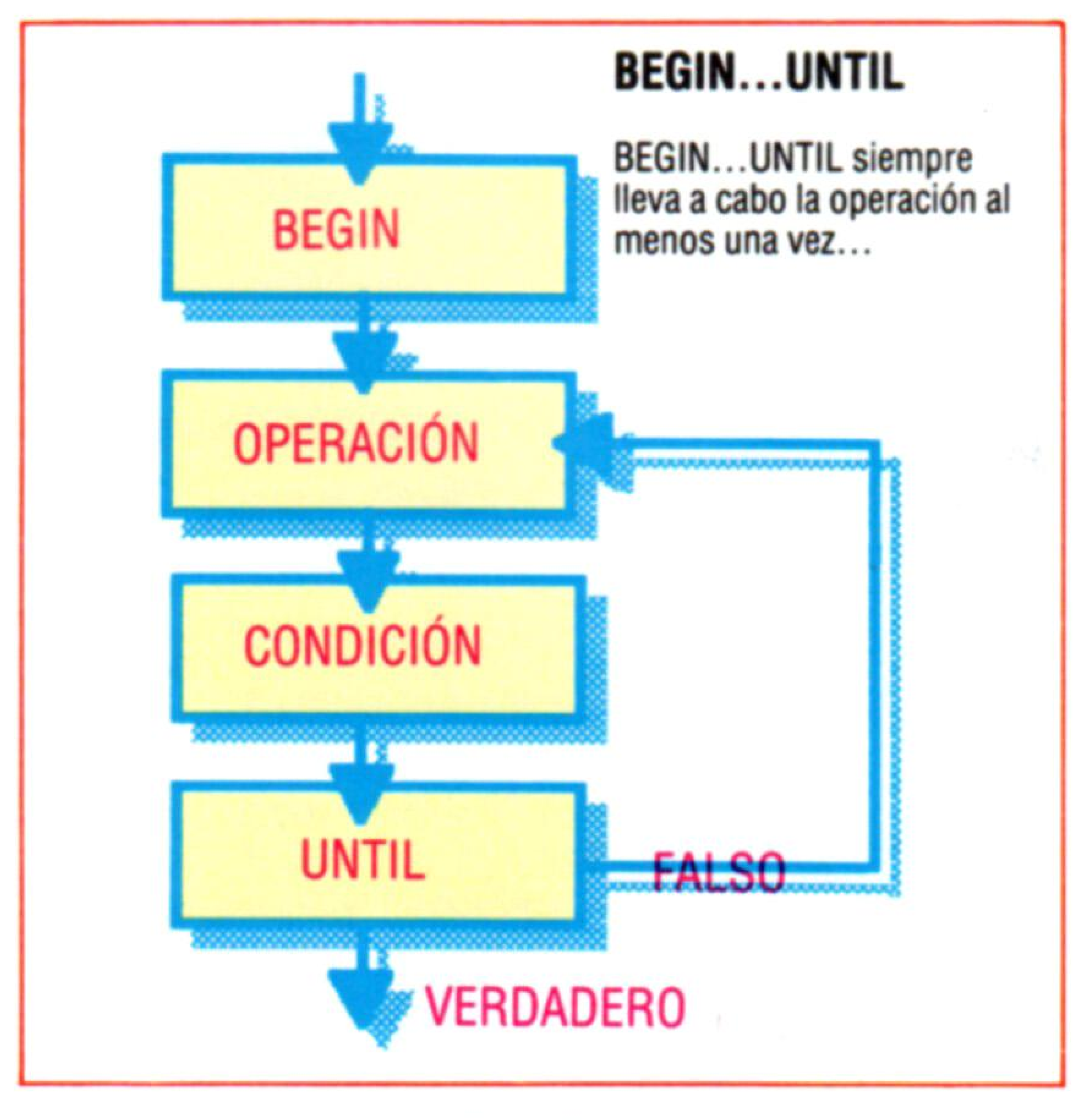
- BEGIN...UNTIL

Es para efectuar un bucle mientras se mantenga una condición. La mayoría de los lenguajes para ordenador modernos, incluyendo los dialectos más nuevos de BASIC, disponen de algo similar. Su formato es:

BEGIN parte bucle condición UNTIL

y ejecuta la parte bucle y la condición; la condición es simplemente el fin de la parte bucle que deja algo en la pila para UNTIL. Por este motivo, la parte bucle se ha de ejecutar al menos una vez. Por ejemplo:

```
: 2POTENCIAS (--)
               (imprime todas las potencias de 2
               menores que 10 000)
  1            (potencia inicial de 2)
  BEGIN
    CR DUP. (imprimir potencia de 2 en una nueva
             línea)
    2* (tomar siguiente potencia de 2)
  DUP 10 000>UNTIL (comprobar si aún no es
             demasiado grande)
  DROP (dejar última potencia de 2)
;
```




- BEGIN...WHILE...REPEAT

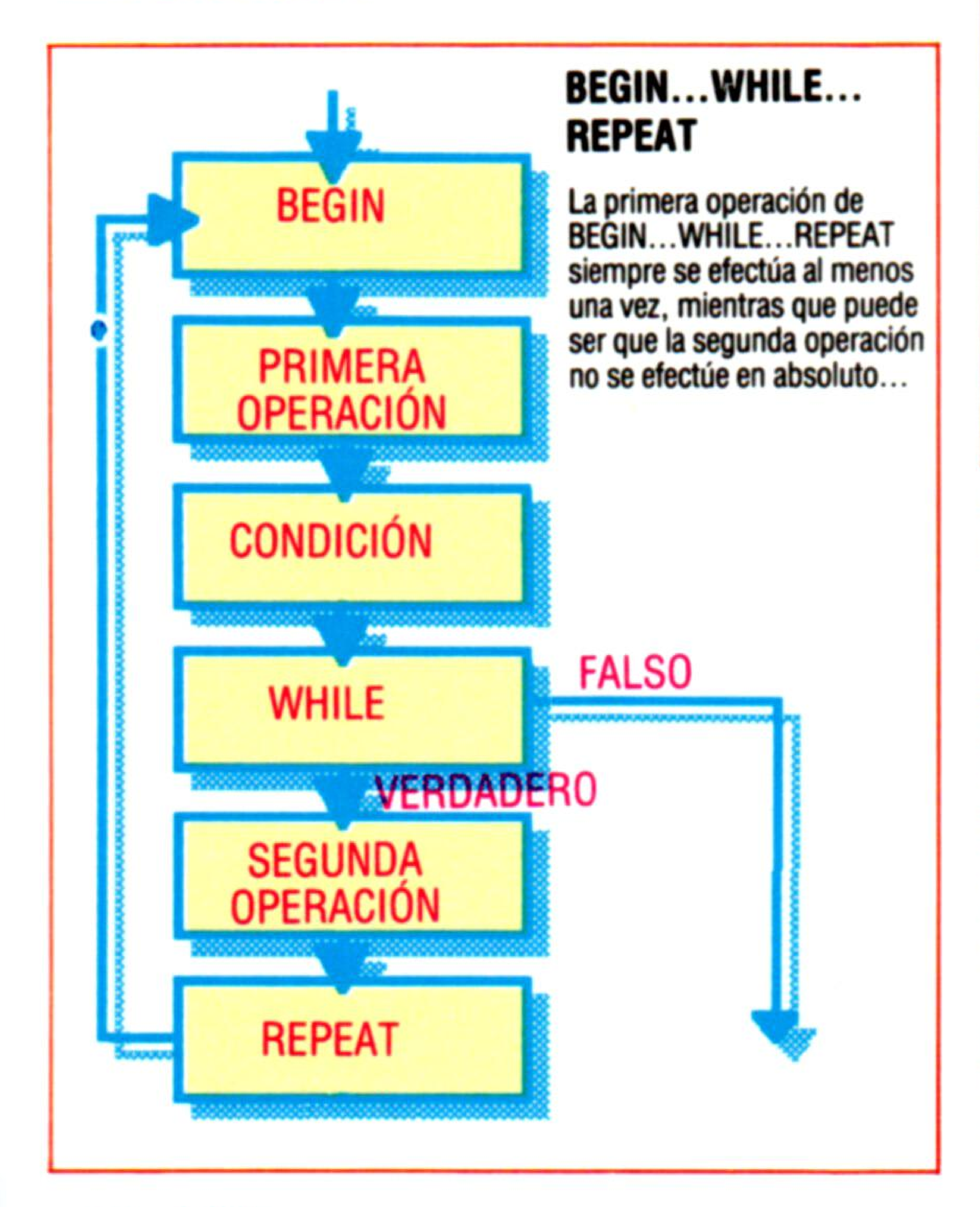
WHILE posee algunas posibilidades más, porque



deja que usted decida, en medio de la parte bucle, si desea salir del bucle sin esperar al fin del mismo. Su formato es:

BEGIN 1.ª parte bucle condición WHILE 2.ª parte bucle REPEAT

Al igual que la parte bucle de UNTIL, aquí la 1.ª parte bucle siempre se ejecuta al menos una vez y termina quedando la condición en la pila. Pero ahora hay dos importantes diferencias respecto a UNTIL. Primero, el bucle ahora se detiene cuando la condición es falsa. Cuando se interrumpe el bucle, el FORTH se salta a la 2.ª parte bucle y continúa a partir del REPEAT. En segundo lugar, si el bucle ha de continuar, el FORTH ejecuta la 2.ª parte bucle antes de volver al bucle desde BEGIN.




**BEGIN...WHILE... REPEAT**

La primera operación de BEGIN...WHILE...REPEAT siempre se efectúa al menos una vez, mientras que puede ser que la segunda operación no se efectúe en absoluto...

- DO...LOOP

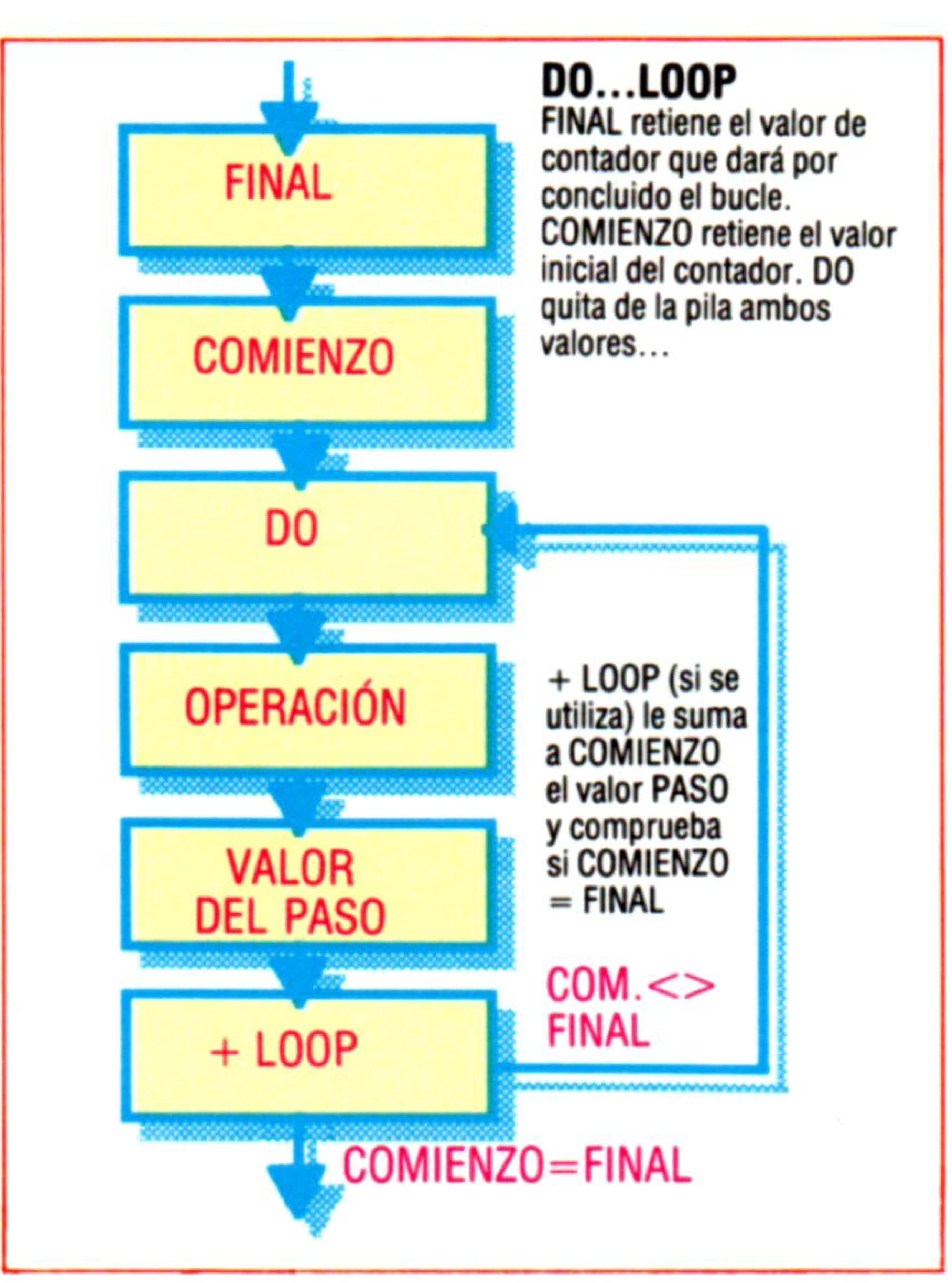
El bucle DO del FORTH hace el mismo trabajo que el bucle FOR del BASIC y muchos otros lenguajes. Comparémoslo con las implementaciones del BASIC:

| BASIC | FORTH |
|---|---|
| FOR X=inicial TO límite | (límite+1) inicial DO |
| cuerpo del bucle | cuerpo del bucle |
| NEXT X | LOOP |
| FOR X=inicial TO límite STEP paso | (límite+1) inicial DO |
| cuerpo del bucle | cuerpo del bucle |
| NEXT X | paso+LOOP |

Existen algunas diferencias evidentes.

En primer lugar, como cabría esperar, el valor inicial y el límite preceden a DO, de modo que DO pueda tomarlos de la pila. El límite va primero. Menos obvio es el hecho de que el límite es uno más el valor final con el cual se desea efectuar el bucle, de modo que 4 0 DO hace el bucle con los valores 0, 1, 2 y 3 (cuatro veces), pero no con 4.

Como en BASIC, el paso (*step*) es opcional. Si usted lo omite y usa LOOP, el paso se toma como 1.




**DO...LOOP**

FINAL retiene el valor de contador que dará por concluido el bucle. COMIENZO retiene el valor inicial del contador. DO quita de la pila ambos valores...

Si lo usa, entonces necesitará +LOOP en vez de LOOP, indicando el paso justo antes de +LOOP.

En BASIC siempre hay una variable de control (X, en el ejemplo anterior). En FORTH no se utiliza ninguna variable como ésta; en cambio, hay una palabra I que coloca en la pila el valor del bucle. Si tiene un bucle DO dentro de otro, entonces I hace referencia al valor del bucle más interior y J hace referencia al siguiente hacia afuera. De haber más bucles DO, luego sus valores de bucle todavía tienen entidad y se cuentan correctamente.

He aquí un ejemplo que eleva un número a la potencia de otro. A diferencia del BASIC, el FORTH no tiene incorporado ningún operador de este tipo; usted define el suyo propio:

```
:  **(m,n--m**n)
  DUP 0 < IF (si n<0 la respuesta se toma como 0)
    DROP DROP 0
  ELSE
```

# Condiciones

Las siguientes palabras son útiles para elaborar condiciones para IF, UNTIL y WHILE. Sus resultados, lo que dejan en la pila, son bien −1 para verdadero, bien 0 para falso, pero sólo en FORTH-83. Los FORTH más antiguos dejan 1 para verdadero y 0 para falso. Ni IF, ni UNTIL ni WHILE insisten en que se les dé 0, 1 o −1; 0 es falso y cualquier otro número es verdadero.

| | |
|---|---|
| = | m,n -- verdadero o falso (verd. si m = n) |
| < | m,n -- verdadero o falso (verd. si m <n) |
| > | m,n -- verdadero o falso (verd. si m > n) |
| 0= | m -- verdadero o falso (verd. si m = 0) |
| 0< | m -- verdadero o falso (verd. si m < 0) |
| 0> | m -- verdadero o falso (verd. si m > 0) |
| NOT | verdadero o falso -- falso o verdadero |
| AND | m,n -- m AND n |
| OR | m,n -- m OR n |

```
    DUP 0=F (si n=0 la respuesta es 1)
      DROP DROP 1
    ELSE
        1 (multiplicará éste por m n veces)
        SWAP 0 DO (para hacer bucle n veces)
          OVER * (mult. parte sup. de pila por m)
        LOOP
        SWAP DROP (dejar m)
      THEN
    THEN
;
```

Los casos especiales son bastante complicados. Primero, si la potencia es negativa, la respuesta correcta es 1 dividido por la respuesta, con la correspondiente potencia positiva. Dado que el FORTH sólo trabaja con enteros, no se puede efectuar esta división exactamente, de modo que es razonable decir que ** devuelve 0 como resultado. El caso especial en el cual n=0 es más sutil. A la vista del mismo, si multiplica 1 por m cero veces (pasando por el bucle cero veces), entonces obtendrá en cualquier caso el resultado 1. Sin embargo, un bucle DO del FORTH siempre se ejecuta al menos una vez y esto arruina el caso en el cual n=0. De hecho, en muchas ocasiones usted no desea necesariamente que el bucle DO se ejecute en un programa, en cuyo caso ha de tomar precauciones especiales.

He aquí otro ejemplo. Se trata de una versión más amplia de 2POTENCIAS, que ofrecimos anteriormente. Utiliza no sólo I, el valor de bucle, sino también la palabra LEAVE, que interrumpe el bucle inmediatamente y continúa desde LOOP o +LOOP:

```
:2 POTENCIAS   (n--)
               (visualiza índices y potencias de
               2 mientras las potencias sean
               menores que n)
15 0 DO (2**14 es el mayor que se puede
   imprimir)
   DUP 2 I**< IF (si 2**I>n)
     LEAVE
   ELSE
     CR I. 2 I**.
   THEN
LOOP
DROP (deja a n)
;
```

Con estas estructuras de programación, usted puede hacerlo sin el GOTO y sin números de línea; en realidad, el FORTH no los posee. Después de haber utilizado el BASIC, le llevará un poco de tiempo habituarse a esto, pero el hecho es que los números de línea son algo impuesto por el BASIC y no constituyen ninguna parte intrínseca de la forma en que uno concibe un programa. Si usted hubiera escrito 2POTENCIAS, ¿se hubiera acordado del DROP del final? Esto ilustra la importancia de ir observando muy de cerca lo que una palabra hace con la pila. Para facilitar las cosas conserve sus definiciones de palabras lo más cortas posible, descomponiendo el problema, como en la segunda definición de 2POTENCIAS. En ese caso, aunque la palabra hace algo más al dejar que usted imponga su límite sobre la pila en lugar de 1000, no por ello es más complicado. Si hubiera pensado en modificar la primera definición para hacer lo mismo, enseguida habría obtenido toda clase de SWAPs y OVERs. La segunda definición es simple porque usted ha dividido el problema definiendo **, lo que le ofrece dos definiciones más cortas en lugar de una.

## Imprimiendo series

Para imprimir una serie desde el interior de una definición de dos puntos, se utiliza ." así:

." serie"

Se necesita al menos un espacio tras el ." porque se trata de una palabra. Si tuviera más de uno, entonces los espacios que siguieran al primero se considerarían como parte de la serie.

La serie sigue al ." (de modo que no está en notación polaca inversa) porque la pila del FORTH no puede manipular series. Las variables en serie de hecho no se proporcionan como estándar en FORTH, pero existen formas de ampliar el lenguaje para manipularlas.

La palabra CR (retorno de carro) envía a la pantalla una nueva línea

## Limitaciones

Si bien las ideas principales tras el DO son bastante directas, también tiene algunas particularidades. En primer lugar, si el paso (antes de +LOOP) es negativo, el valor del bucle realmente puede alcanzar el límite en vez de detenerse justo antes del mismo. De modo que 2 0 DO ... –1 +LOOP efectúa el bucle a través de los valores 0, –1 y –2.

En segundo lugar, hemos visto que un bucle DO siempre se ejecuta al menos una vez, aun para 0 0 DO...LOOP, que quizá usted quisiera saltarse. El FORTH-83 lo sorprenderá ejecutando el bucle 65 536 veces. –1 0 DO...LOOP lo hará 65 535 veces, –2 0 DO...LOOP lo hará 65 534 veces, y así sucesivamente.

La respuesta más simple no es accidental, sino que tiene sentido si se es consciente de la forma en que se pueden tratar los enteros del ordenador, ya sea con signo o sin signo (complemento de a dos). Si el valor inicial ya ha pasado el límite, entonces subirá hasta 32 768, que es tratado como equivalente de –32 768, y después seguirá hacia arriba a través de los enteros negativos hasta llegar al límite desde abajo. En los FORTH más antiguos, si el valor inicial ya ha superado el límite, el bucle se efectúa sólo una vez. Asimismo, los FORTH más antiguos tratan el LEAVE de forma algo distinta. No saltan directamente fuera del bucle, sino que se aseguran de que se efectuará una salida la próxima vez que se encuentre LOOP o +LOOP



# Datos básicos (VI)

**Proseguimos el análisis del mapa de memoria del C64, por cortesía de la Commodore Business Machines**

| ETIQUETA | DIRECCIÓN HEXA | POSICIÓN DECIMAL | DESCRIPCIÓN |
|---|---|---|---|
| PNTR | 00D3 | 211 | Columna cursor en la línea actual |
| QTSW | 00D4 | 212 | Flag: editor en modo comillas, $00 = NO |
| LNMX | 00D5 | 213 | Longitud de línea física de la pantalla |
| TBLX | 00D6 | 214 | Número línea física cursor actual |
| | 00D7 | 215 | Área temp. de datos |
| INSRT | 00D8 | 216 | Flag: Modo insert, >0 = #INSTs |
| LDTB1 | 00D9-00F2 | 217-242 | Tabla enlace línea pantalla/editor temp. |
| USER | 00F3-00F4 | 243-244 | Puntero: pos. RAM color actual pantalla |
| KEYTAB | 00F5-00F6 | 245-246 | Vector: tabla decodificación teclado |
| RIBUF | 00F7-00F8 | 247-248 | Puntero buffer entrada del RS232 |
| ROBUF | 00F9-00FA | 249-250 | Puntero buffer salida del RS232 |
| FREKZP | 00FB-00FE | 251-254 | Espacio página 0 libre para programas usuario |
| BASZPT | 00FF | 255 | Área datos temp. del BASIC |
| | 0100-01FF | 256-511 | Área pila del sistema microprocesador |
| | 0100-010A | 256-266 | Área trabajo flotante para cadenas |
| BAD | 0100-013E | 256-318 | *Log* para error entrada cinta |
| BUF | 0200-0258 | 512-600 | Buffer de INPUT sistema |
| LAT | 0259-0262 | 601-610 | Tabla núcleo: números archivo lógico activo |
| FAT | 0263-026C | 611-620 | Tabla núcleo: número dispositivo para cada archivo |
| SAT | 026D-0276 | 621-630 | Tabla núcleo: segunda dirección para cada archivo |
| KEYD | 0277-0280 | 631-640 | Cola buffer teclado (FIFO: *first in, first out:* primero en entrar, primero en salir) |




Crispin Thomas

**Acertadas mejoras**
El Amstrad CPC 664 es una versión mejorada del muy popular micro CPC 464. En lugar de una platina de cassette, esta máquina lleva incorporada una unidad de disco estándar Hitachi de 3 pulgadas. Se han agrandado las teclas del cursor para una mayor facilidad de uso y, además, Amstrad ha mejorado la ROM de BASIC para incluir diez instrucciones adicionales, así como las instrucciones del DOS

# Sucesor del CPC 464

**El nuevo ordenador presentado por Amstrad, el CPC 664, cuenta con una unidad de disco incorporada y un diseño integral, que se suman a los méritos de su antecesor, el CPC 464**

El ordenador personal Amstrad CPC 464 fue muy elogiado cuando se lanzó en 1984. Aunque no representaba ninguna innovación desde el punto de vista técnico, la máquina constituyó un éxito, al que contribuyeron, sin duda alguna, sus excelentes gráficos y su fiable hardware. Un año después, la empresa lanzaba una segunda máquina, el Amstrad CPC 664, esencialmente similar a la anterior pero con una unidad de disco incorporada en lugar de la platina de cassette.

El trazado del CPC 664 es idéntico al del anterior CPC 464, ahora con algunas teclas de color celeste en vez de verdes. Amstrad también ha decidido numerar las teclas del relleno numérico F1, F2, y así sucesivamente, si bien las teclas cumplen exactamente las mismas funciones. La otra diferencia entre los dos teclados reside en el mayor tamaño de las teclas del racimo del cursor, que permiten manipular el cursor con mayor facilidad.

Con una unidad de disco incorporada, obviamente han desaparecido las teclas de la cassette, y en su lugar hay ahora una unidad de Hitachi estándar de 3 pulgadas. La unidad de disco parece mucho más pequeña que la unidad de disco externa, porque la unidad del CPC 664 funciona con una fuente de alimentación de 12 V de la pantalla que viene con el ordenador.

Dado que el CPC 464 tenía su propia platina de cassette incorporada, no había necesidad de incluir una puerta para cassette; pero en el 664 sí se ha añadido esta facilidad, para que los usuarios puedan aprovechar la base de software que hayan ido creando en cinta para la máquina más antigua.

El Amstrad CPC 464 posee un único bus de ampliación destinado no sólo a la adición de una unidad de disco flexible, sino también como interface para periféricos de fines generales. El Amstrad 664 no sólo tiene una interface para periféricos, sino además una puerta adicional para una segunda unidad de disco flexible. Ésta es una facilidad importante si la máquina ha de utilizar CP/M, porque muchos paquetes escritos para trabajar bajo este sistema operativo exigen que el disco de aplicaciones esté en una unidad y el disco de datos en otra. Por supuesto, se puede ejecutar CP/M disponiendo de una sola unidad, pero esto por lo general supone tener que ir cambiando los discos.

Al producir la nueva máquina, Amstrad ha tenido oportunidad de mejorar la ROM de BASIC para que incluya algunas instrucciones nuevas. Están, por supuesto, las instrucciones del sistema operativo de disco, que son idénticas a aquellas incorporadas en la DDI ROM de la interface para disco flexible externo. Muchas de las nuevas instrucciones añaden configuraciones a la ya poderosa lista de instrucciones para gráficos de que dispone el programador de BASIC. Probablemente la más importante de ellas sea la instrucción FILL, una curiosa omisión en el BASIC original. Esta instrucción rellenará una superficie ya sea con el color de primer plano actual o bien con el color que determine el programador.



# Instrucciones para gráficos

El programador de gráficos dispone asimismo de la instrucción MASK, que puede producir una línea de puntos. La instrucción se escribe con el formato MASK i,p, donde i es un entero entre 1 y 255, y p determina si se debe trazar o no el primer punto. Como su nombre sugiere, la instrucción es una «máscara» que lleva a cabo una operación lógica de gestión sobre la celda de caracteres. De modo que MASK 1,p producirá un solo punto cada ocho pixels, mientras que MASK 17,p producirá dos puntos. Establecida en 255, MASK presentará una línea continua.

GRAPHICS PEN y GRAPHICS PAPER son dos instrucciones relacionadas con MASK. Normalmente, cuando dibujamos una línea no podemos ver el color de fondo. Sin embargo, cuando se traza una línea de puntos con la instrucción MASK, es deseable ver el color del fondo. De modo que GRAPHICS PAPER nos permite establecer los gráficos del fondo ya sea en el color PAPER actual o bien en algún otro color. Del mismo modo, GRAPHICS PEN establece el color de fondo para las líneas o los puntos.

Para refinar aún más el proceso de escribir gráficos en la pantalla, se ha añadido una instrucción adicional, FRAME. Cuando se producen gráficos, se suele observar en la pantalla cierto grado de parpadeo. Ello se debe a que los gráficos están intentando aparecer en la pantalla en medio de una exploración.

El efecto de la instrucción FRAME es interrumpir la ejecución del programa en BASIC hasta que la exploración recomience en la parte superior de la pantalla, de modo que se puedan incluir los gráficos en un solo cuadro. Esta instrucción, por consiguiente, produce una visualización mucho más uniforme.

**Control de instrucciones**
Esta instantánea de pantalla muestra tres de las nuevas instrucciones de que dispone el Amstrad CPC 664. Las líneas de puntos del fondo se dibujan pulsando MASK, mientras que los puntos de la estrella se trazan mediante DRAW (que dibuja en una posición relativa a la posición actual del cursor). Por último, las superficies del interior de la estrella se colorean mediante FILL

Para simplificar la manipulación de datos en la pantalla se ha añadido COPY CHR$. Esta instrucción significa que los caracteres de una parte de la pantalla se pueden transferir a otra zona. Esto es útil para aplicaciones de gestión, en las que, p. ej., se puede trasladar a cualquier parte una lista de cifras o de direcciones de una ventana con el fin de procesarlas.

Éstos son apenas unos ejemplos de las numerosas instrucciones que se han añadido al BASIC. Aunque se pretende que la ROM sea compatible con la versión de BASIC anterior, ciertos paquetes diseñados para ejecutarse en el CPC 464 no funcionarán en el 664. Ello no se debe a un error de Amstrad. Al igual que otros muchos fabricantes, la empresa se ha reservado el derecho de mejorar el BASIC cuando lo crea necesario. Con el fin de mantener la compatibilidad, la empresa incluyó un bloque de saltos para vectorizar las llamadas a direcciones en ROM.
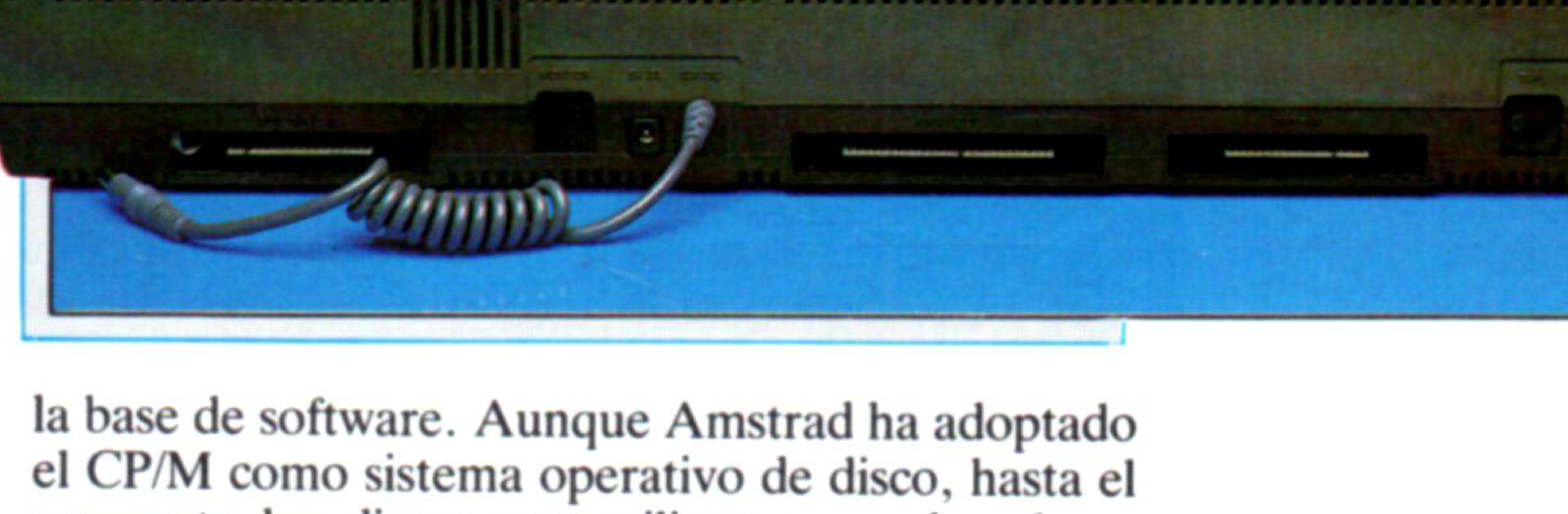
Lamentablemente, al objeto de proporcionar velocidad adicional, algunos fabricantes de software independientes han pasado por alto el bloque de saltos, llamando directamente a las rutinas. Debido a que en la nueva versión de la ROM muchas de estas rutinas han cambiado de dirección, una gran parte del software ya no servirá.

El Amstrad CPC 664 está destinado a ser un ordenador tanto para pequeña gestión como para aficionados personales. Como tal, parece ofrecer excelentes prestaciones. No obstante, quedan pendientes uno o dos interrogantes acerca de la viabilidad de la nueva máquina como ordenador de gestión. En primer lugar, está el perenne fantasma de la base de software. Aunque Amstrad ha adoptado el CP/M como sistema operativo de disco, hasta el momento los discos que utilizan sus ordenadores no han conseguido hacerse de una gran popularidad. De hecho, esto significa que normalmente los clientes tendrán que esperar a que aparezca el software en el formato correcto.

Sin embargo, gracias a un acuerdo establecido entre Amstrad y la empresa reproductora Timatic, también se pueden copiar discos CP/M de formato de 5 1/4 pulgadas. También existe la posibilidad, si fuera necesario, de adquirir una unidad de disco adecuada de algún proveedor independiente, para usar conjuntamente con la unidad Hitachi.

Algunos de los paquetes CP/M estándares, como el *WordStar* o el *dBase II*, no se ejecutarán en el Amstrad. Ello se debe a que el ordenador sólo tiene 39 Kbytes libres para programas de aplicaciones cuando opera bajo CP/M. No obstante, se ha anunciado la aparición de una placa de ampliación de memoria para el ordenador, que permitirá que los usuarios saquen el máximo partido de la enorme base de datos CP/M.

**Parte posterior**
Aunque las interfaces de la parte posterior son similares a las del CPC 464, se han efectuado algunas adiciones. A la derecha se ha añadido una puerta para cassette, mientras que en el lado izquierdo se ha incluido una interface para unidad de disco flexible, lo que deja libre el bus de ampliación para otros usos. El cable unido al ordenador se alimenta con la pantalla para operar la unidad de disco incorporada




**AMSTRAD CPC 664**

INTERFACES

Bus de ampliación, interface para segunda unidad de disco flexible, puerta para cassette, puerta E/S, interface para palanca de mando, interface para impresora, entrada 12 V, entrada 5 V, conector para pantalla

SOFTWARE SUMINISTRADO

El CPC 664 viene con CP/M 2.2 y Dr LOGO

DOCUMENTACION

El manual contiene detalladas explicaciones sobre el BASIC Amsoft, Dr LOGO y el CP/M, y es una recopilación de los manuales del CPC 464 y del disco flexible DDI

PUNTOS FUERTES

Atendiendo a su precio, es difícil encontrarle defectos al CPC 664 como máquina de pequeña gestión. Hace unos pocos años, un sistema similar habría costado varias veces su precio

PUNTOS DEBILES:

Hasta que se rectifique con la introducción de una placa para ampliación de memoria, el ordenador sufrirá de carencia de espacio de memoria disponible para ejecutar los paquetes CP/M utilizados más comúnmente

# Preparando el escenario

## El diseño de juegos de aventuras exige un enfoque sumamente estructurado

El programa que escribiremos para proporcionarle un entorno a nuestra rutina de personajes interactivos tendrá la estructura que se refleja en el diagrama. Iremos trabajando el programa paso a paso, comenzando por las rutinas de inicialización. Para simplificar, utilizaremos matrices en serie para almacenar todos los datos y leeremos los valores por defecto en las matrices desde sentencias de datos.

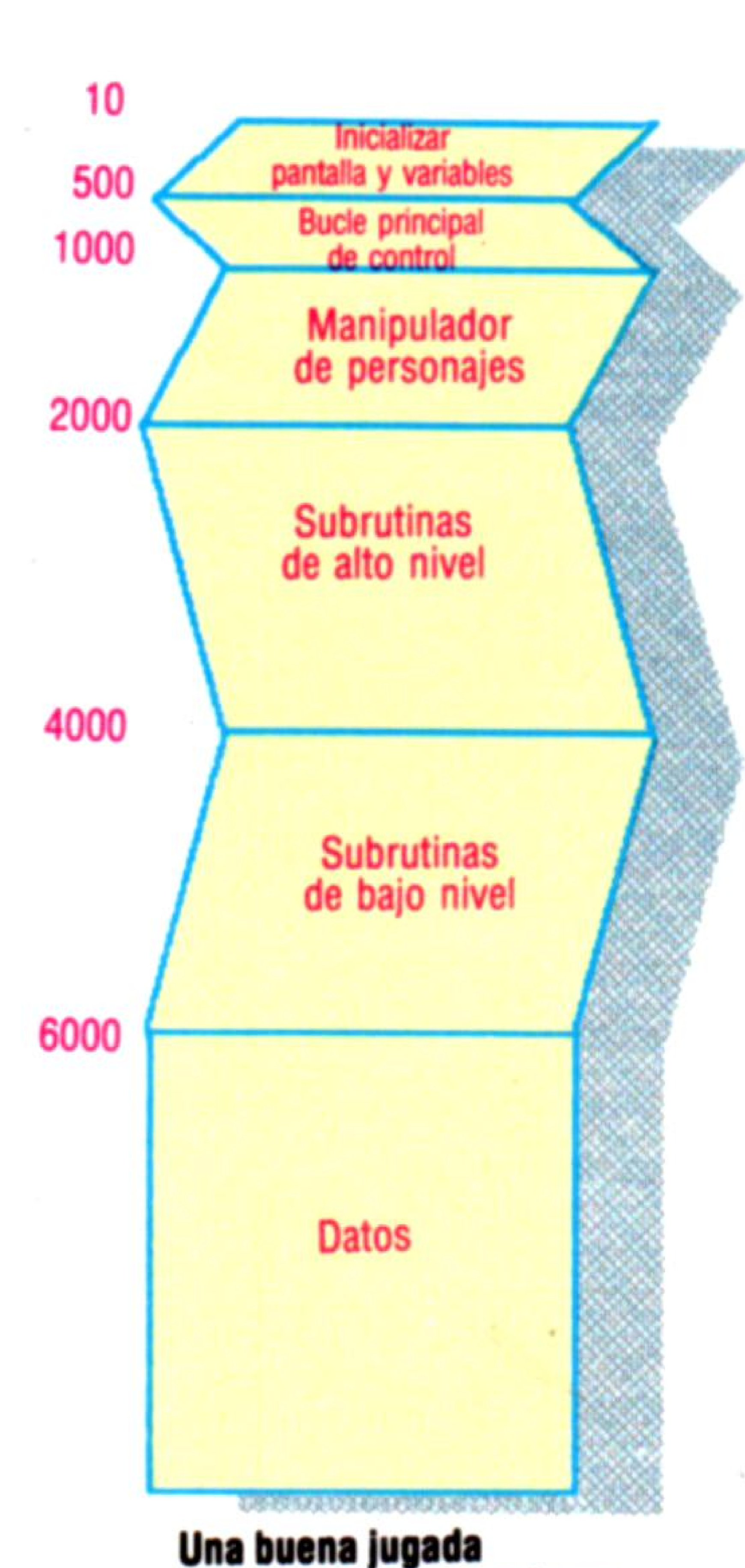
Primero y principal, necesitamos tres descripciones de escenarios: una para cada habitación del concurrido *pub* Dog and Bucket (El perro y el cubo). Asimismo, necesitamos algún medio de saber cómo están comunicadas las habitaciones, de modo que si un personaje avanza hacia el este desde la sala de tertulia, por ejemplo, sepamos que terminará en el salón. La matriz bidimensional l$ retiene toda esta información de la forma indicada en nuestra *Tabla de matrices*.

La decisión de cómo representar los datos para los objetos de nuestro juego dependerá básicamente del papel que se espera desempeñen. En este juego, nos importan los objetos como elementos para que los personajes recojan, dejen o, quizá, se arrojen los unos a los otros. En el caso de las empanadas y el bocadillo de jamón, también queremos considerar la cuestión de si son o no comestibles; y, por supuesto, ¡que haya muchísimo para beber! Por lo tanto, es necesario que las estructuras de datos de objetos sean capaces de abordar las siguientes preguntas de nuestro juego: ¿dónde está, es comestible, es bebestible?

Para hacerlo, utilizamos la matriz bidimensional b$, tal como se indica en la *Tabla de matrices*. Durante el programa podremos comprobar los elementos de esta matriz en cualquier momento, para responder a cualquiera de las tres preguntas que acabamos de enumerar.

Los objetos y los escenarios plantean problemas, de modo que encaremos la tarea, más interesante, de decidir cómo almacenar la información sobre los personajes.

El Dog and Bucket da cobijo a los siguientes alegres parrandistas: Luis Cubas, Lola Fiestas, Pepe Viñas, Mari Tapas, Javi Salado y Gina Fizz. También incluiremos otro personaje: Fred, el barman. Este último personaje no es interactivo; el barman se incluye simplemente en una descripción de escenario y el manipulador de personajes ocasionalmente visualizará mensajes sobre sus «acciones». De esta forma, vemos que un personaje, para ser eficaz, no requiere necesariamente una programación y una manipulación complejas durante el tiempo de ejecución.




**Una buena jugada**
La adopción de un enfoque modular a la programación nos permitirá modificar nuestro juego en una etapa ulterior si así lo deseáramos, así como adaptar el manipulador de personajes para ejecutarlo con otros programas menos comprometidos. Como en la mayor parte del software de aventuras, el grueso de la memoria lo ocupan los datos, la mayoría de los cuales serán mensajes de texto para imprimir en la pantalla

### Tabla de objetos

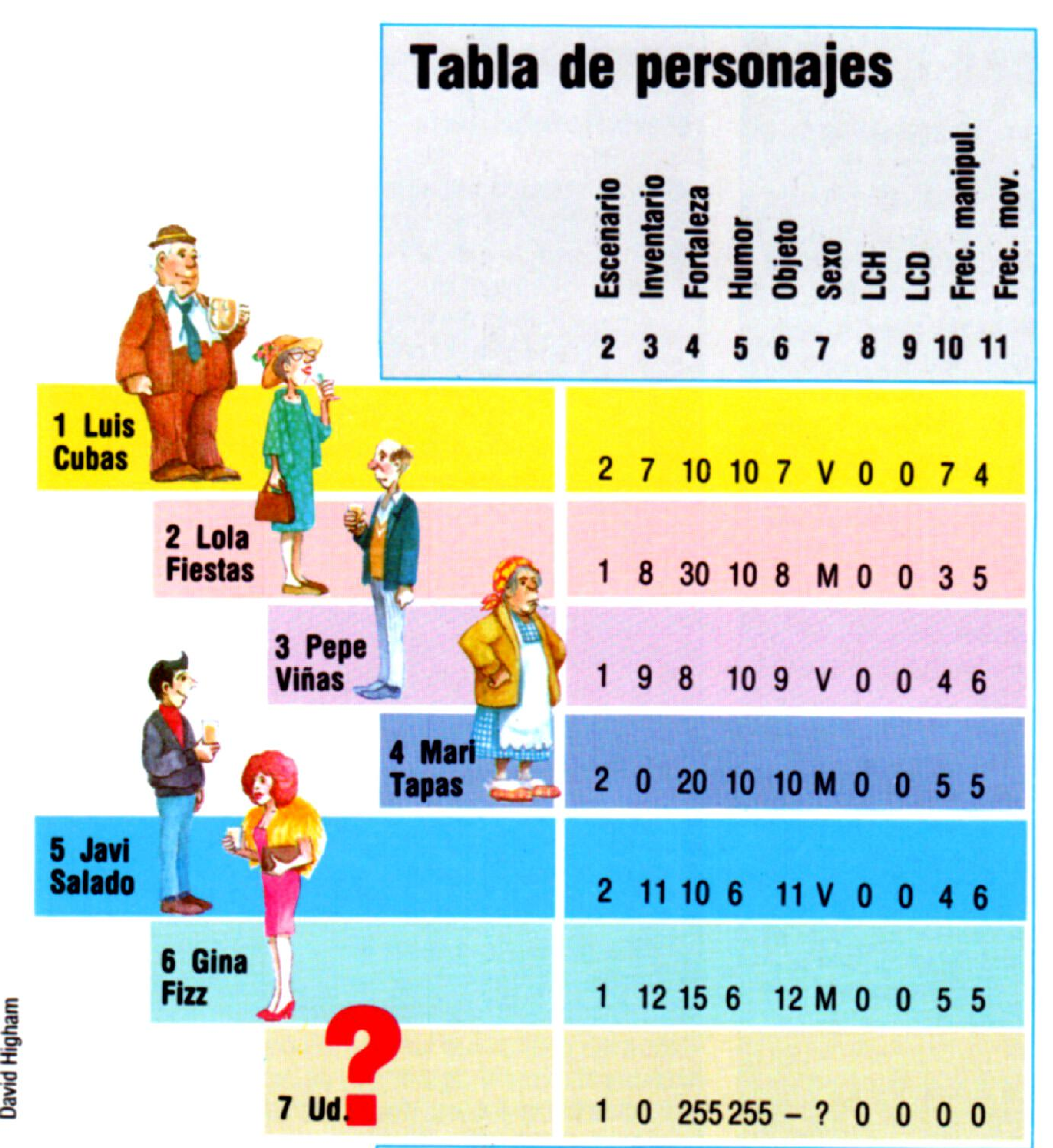
| Número objeto | Descripción | Escenario inicial | ¿Comestible? | ¿Bebestible? |
|---|---|---|---|---|
| 1 | Un vaso de cerveza | 2 | N | S |
| 2 | Una lata vacía de comida para gatos | 3 | N | N |
| 3 | Empanada de la casa | 1 | S | N |
| 4 | Una banqueta de bar | 2 | N | N |
| 5 | Un cenicero | 1 | N | N |
| 6 | Un bocadillo de jamón rancio | 2 | S | N |
| 7 | Una pinta de cerveza amarga | 0 | N | S |
| 8 | Una crema de menta | 0 | N | S |
| 9 | Un whisky con soda | 0 | N | S |
| 10 | Un vodka puro | 2 | N | S |
| 11 | Una pinta de cerveza añeja | 0 | N | S |
| 12 | Una ginebra con ginger ale | 0 | N | S |

**Lección objetiva**
En el Dog and Bucket se pueden encontrar 12 objetos diferentes. Observe que los elementos del 7 al 12 son «propiedad» de los personajes (ver *Tabla de personajes*) y, por tanto, con la excepción del número 10, parten desde el escenario 0, ya que los llevan consigo sus dueños. Al comienzo, sin embargo, Mari Tapas ha extraviado su trago (número 10), el cual se halla en el salón

Para decidir los atributos que necesitamos almacenar para nuestros personajes, recordemos primero la lista de atributos que considerábamos necesarios para una "persona" controlada por ordenador. Primero, el desplazamiento de un escenario a otro es esencial, por lo que es obvio que necesitamos llevar el registro de la posición de un personaje. Segundo, los personajes han de ser capaces de manipular objetos, por lo que en algún lugar a lo largo de la línea se debe incluir un registro del inventario de cada personaje.

Los otros atributos fundamentales de un personaje están relacionados con la comunicación con el jugador y su conciencia del entorno. En realidad, no es necesario implementar el primer aspecto en la rutina manipuladora de personajes propiamente dicha. Para entender por qué, piense tan sólo en lo que sucede cada vez que usted entra una instrucción en un juego. Si digita, por ejemplo, Coger daga, y el arma está presente, el programa corregirá

## Tabla de personajes



| | Escenario | Inventario | Fortaleza | Humor | Objeto | Sexo | LCH | LCD | Frec. manipul. | Frec. mov. |
|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 Luis Cubas | 2 | 7 | 10 | 10 | 7 | V | 0 | 0 | 7 | 4 |
| 2 Lola Fiestas | 1 | 8 | 30 | 10 | 8 | M | 0 | 0 | 3 | 5 |
| 3 Pepe Viñas | 1 | 9 | 8 | 10 | 9 | V | 0 | 0 | 4 | 6 |
| 4 Mari Tapas | 2 | 0 | 20 | 10 | 10 | M | 0 | 0 | 5 | 5 |
| 5 Javi Salado | 2 | 11 | 10 | 6 | 11 | V | 0 | 0 | 4 | 6 |
| 6 Gina Fizz | 1 | 12 | 15 | 6 | 12 | M | 0 | 0 | 5 | 5 |
| 7 Ud. | 1 | 0 | 255 | 255 | – | ? | 0 | 0 | 0 | 0 |

**Dramatis personae**
Nuestra tabla refleja los valores iniciales de los diferentes atributos para cada uno de los personajes del Dog and Bucket. Todos los personajes empiezan con sus propios tragos en su inventario, con excepción de Mari Tapas. El personaje número 7 representa al jugador, ¡y se ha incluido para asegurar que los otros personajes le "presten atención" a usted! No obstante, el factor de manipulación del personaje 7 es cero, lo que asegura que sus acciones serán de su entera responsabilidad y no estarán sujetas a la rutina manipuladora de personajes

David Higham

la variable que retiene la posición de la daga para indicar que ahora se la está transportando, y luego alterará en consecuencia el inventario del jugador. Por el contrario, si usted entra una instrucción para que otro personaje tome un objeto, el programa simplemente lleva a cabo el mismo proceso, pero en esta ocasión altera el inventario del personaje en cuestión.

En este punto no existe necesidad alguna de implicar a la rutina manipuladora de personajes: el manipulador existe sólo para asegurar que éstos puedan llevar su propia vida con total independencia del jugador. Por supuesto, cuando se ejecute el manipulador de personajes, encontrará que ahora la daga está en el inventario del personaje y actuará en consecuencia.

La forma más simple de asegurar que los personajes y el jugador se lleven bien quizá sea también la más obvia: que usted se implemente a sí mismo como una persona más. Todo cuanto debe hacer es definir un personaje llamado Ud., y cada vez que se llame a la rutina manipuladora de personajes, se encontrará a sí mismo profundamente inmerso en las vidas de sus compañeros de ficción.

La comunicación con el jugador no es en principio difícil de comprender ni de implementar eficazmente en el programa (si bien tal comunicación siempre resulta bastante limitada), pero incluir la conciencia del entorno (el último requisito previo para un personaje interactivo) puede resultar muy complicado. En un grado limitado, se puede incluir esta configuración en nuestro manipulador de personajes simplemente sobre la base de los datos que ya hemos listado. La rutina puede comprobar si hay objetos presentes, manipularlos o incluso generar algún comentario inteligente sobre ellos que sea "pronunciado" por un personaje. Asimismo, puede comprobar la posición de un personaje y acaso hacer que éste haga algún comentario sobre la misma. Sin embargo, aunque útiles, las rutinas como éstas son más bien limitadas. Obviamente, lo que se requiere es una conciencia sobre los otros personajes y, en particular, la capacidad de otorgar a una «persona» controlada por ordenador algún tipo de «continuidad», es decir, proporcionar a los personajes un sentido de la historia, del cual ya hemos hablado anteriormente en esta serie.

Para hacer esto introduciremos dos nuevos atributos a almacenar para cada personaje: el código de última instrucción (LCD) y el código de último personaje (LCH). Éstos indican, respectivamente, la última acción efectuada por el personaje (o, en algunos casos, la última infligida a él), y la otra parte (si la hubiera) involucrada en esta acción. Estos atributos se pueden incluir en la matriz de datos de nuestros personajes y, a modo de ejemplo de la forma en que se podrían utilizar, consideremos el caso en el cual Luis Cubas (personaje número 1) le da una empanada a Lola Fiestas (personaje número 2).

El LCD de esta última ahora se establecerá para indicar una acción "recibir", y su LCH retendrá el número 1. La próxima vez que se llame a la rutina manipuladora, ésta comprobará los datos y podrá deducir a partir del objeto retenido en el inventario de Lola (la empanada) qué es lo que acaba de suceder. La rutina puede, entonces, decidir si Lola debe o no decir Gracias; si lo hace, se establecerá su LCD de modo que indique "comunicar" y su LCH seguirá reteniendo 1.

Si, por el contrario, Lola no emprende ninguna acción tras la entrega de Luis, tanto su LCD como su LCH se establecerán en 0, indicando que el pasado, efectivamente, se ha olvidado. El uso del LCD y el LCH puede hacerse muy complejo, como veremos más adelante en la serie cuando lo examinemos con mayor detalle. Mientras tanto, los códigos que utilizaremos para LCD se incluyen en la *Tabla de matrices*.

## Seis atributos

Hay otros seis atributos que almacenaremos para nuestros personajes, cada uno de ellos útil y muy directo:

• *Fortaleza.* Este atributo determina la capacidad del personaje para moverse y comunicarse. En este caso en particular, se tratará a un personaje como si estuviera inconsciente cuando su fortaleza sea igual

## Tabla de matrices

| Matriz | Dimensionada en | Retiene inform. relativa a |
|---|---|---|
| I$ | 3,5 (Spectrum: 3,5,255) | Descr. del escenario (I$(n1)) y los 4 códigos de salida (I$(n,2...5)) |
| b$ | 12,4 (Spectrum: 12,4,30) | Descr. del objeto (b$(n,1)) y escenario/comestible/ bebestible (b$(n,2,...4) |
| c$ | 7,11 (Spectrum: 7,11,15) | Nombres de personajes (c$(n,1)) y sus atributos (c$(n,2,...11)) |

**Códigos de última instrucción**

| LCD | Indica |
|---|---|
| 1 | La última vez que se llamó al manipulador de personajes, el personaje recibió del personaje que indica el LCH el objeto de su inventario |
| 2 | Previamente el personaje dijo algo acerca del personaje que indica el LCH |
| 3 | Monólogo. El personaje acaba de hacer un comentario. Este código se utiliza para impedir que los personajes hagan los mismos o similares comentarios |
| 4 | Tomar. Durante la última ejecución del manipulador de personajes se recogió el objeto del inventario del personaje |
| 5 | Golpear. El personaje acaba de ser golpeado por un objeto que le arrojó otro personaje |
| 6 | Comer. El personaje está comiendo el objeto de su inventario |
| 7 | Entrar. El personaje acaba de entrar en el escenario actual |

**Patrón de retención**

Las tres matrices principales (I$, b$ y c$) retienen todos los datos necesarios relativos a escenarios, objetos y personajes. Observe que no se utilizan elementos cero (para facilitar la compatibilidad entre los diferentes BASIC, algunos de los cuales no disponen de una facilidad elemento cero). Se emplearán matrices en serie para retener datos tanto alfanuméricos como numéricos (usando STR$ y VAL$). El manipulador de personajes utiliza los códigos de última instrucción (LCD) para conferirles a los personajes algún pequeño grado de continuidad o «historia», y están retenidos en c$(n,9) (ver *Tabla de personajes*).

## Tabla de escenarios

| Escenario | Descripción | N S E O |
|---|---|---|
| 1 | Usted se halla en la sala de tertulia del Dog and Bucket. En un rincón hay varios personajes dudosos jugando al dominó. Detrás del mostrador está Fred, el barman, con su habitual aspecto festivo. Hay una puerta que da al este. | 0 0 2 0 |
| 2 | He aquí el salón del pub, al cual le vendría muy bien una decoración total. Parece que el suelo ha sido regado regularmente con cerveza derramada. Hay puertas hacia el oeste y hacia el sur. | 0 3 0 1 |
| 3 | ¡Uf! Ésta es la cocina, donde se preparan las famosas empanadas de carne de la casa para una clientela que siempre tiene apetito. Hay numerosas latas vacías de comida para gatos, lo que no deja de ser extraño, ya que no hay gato | 2 0 0 0 |

**Salida, plataforma oeste**

Cada uno de los tres escenarios posee su propia descripción junto con los cuatro códigos de salida, uno para cada punto cardinal. El código de salida retiene el número de destino; de modo que, p. ej., la salida hacia el norte desde la cocina conduce al escenario 2, el salón. El número cero indica que el movimiento en esa dirección es imposible

o menor que cero. Ciertas acciones (como beber demasiado) reducirán la fortaleza de un personaje, mientras otras, por el contrario, la incrementan.

- *Humor.* Determina la forma en que los personajes se tratan entre sí. Por ejemplo, si la rutina manipuladora encuentra que un personaje está transportando un objeto determinado, quizá llegue a un punto en el cual tenga que decidir si es necesario abandonar el objeto o conservarlo. Si el humor del personaje es particularmente escaso (supongamos, de casi cero) el manipulador acaso determine que el objeto deba arrojarse contra alguien. Entre los factores que influyen sobre el estado de ánimo en el Dog and Bucket se incluye el extraviar el trago que se estaba bebiendo.
- *Objeto.* Cada personaje del programa tiene su objeto "propio", registrando este atributo el número de cada uno. En este caso particular, los objetos son todos bebidas de diversas descripciones, y perder o encontrar el trago de uno tiene ciertos efectos sobre la conducta del personaje.
- *Sexo.* Este atributo registra si un personaje es masculino o femenino. Se utiliza, por ejemplo, para determinar los pronombres correctos a emplear en los mensajes.
- *Frecuencia de manipulación.* La frecuencia de manipulación (HF) determina el tiempo que debe transcurrir antes de que la rutina manipuladora de personajes compruebe al personaje. Por ejemplo, sobre un personaje con una HF de 5 se actuará una vez de cada cinco que se llame a la rutina. Un HF de 0, sin embargo, significará que la rutina no pondrá al día a ese personaje en absoluto. Junto con el atributo de frecuencia de movimiento (del que hablamos a continuación), la HF se puede utilizar para «congelar» los personajes (estableciendo HF en 0), demorarlos (aumentando HF) o darles el aspecto de ser más activos (disminuyendo HF).
- *Frecuencia de movimiento.* Quizá deseemos que ciertos personajes sean activos en relación a sus compañeros y su entorno, pero al mismo tiempo necesitemos que se mantengan en un escenario o asegurarnos de que no se vayan desplazando con demasiada rapidez. La frecuencia de movimiento determina la frecuencia con la que la rutina manipuladora intentará desplazar al personaje.

En el diagrama de la página anterior podemos ver los valores iniciales para cada uno de los atributos de los personajes.

# Alta estrategia

## Diseñaremos una rutina que permita al ordenador realizar movimientos estratégicos

Probablemente el *go* sea uno de los juegos de tablero más difíciles de informatizar. Las dimensiones del tablero y la flexibilidad de los movimientos restringen severamente el uso de árboles de juego anticipados, habituales en la mayoría de los juegos de pericia computerizados, incluyendo el ajedrez, el backgammon y las damas.

En un juego como el ajedrez, donde el número posible de movimientos es relativamente limitado, de una media de alrededor de 30 desde cualquier posición dada, es factible examinarlos a todos de forma detallada, asegurando, de ese modo, que el ordenador encontrará al menos un movimiento para efectuar, ¡aun cuando el movimiento consista en abandonar, devolviendo el rey a su sitio! Lamentablemente, nuestro sistema de evaluación para el *go* no puede trabajar de forma eficiente de esta manera. Nuestra rutina de evaluación de grupos sólo comprobará los movimientos junto a grupos de menos de tres licencias. Si en el tablero no hubiera ningún grupo en esta situación, la rutina no hallaría ningún movimiento. En futuros capítulos crearemos otras rutinas de evaluación basadas en técnicas simples de emparejamiento de patrones, que también hallarán movimientos sólo en ciertas situaciones. Lo que realmente necesitamos es una evaluación «general», que con toda seguridad siempre hallará al menos un movimiento posible, en el supuesto de que haya alguno disponible.

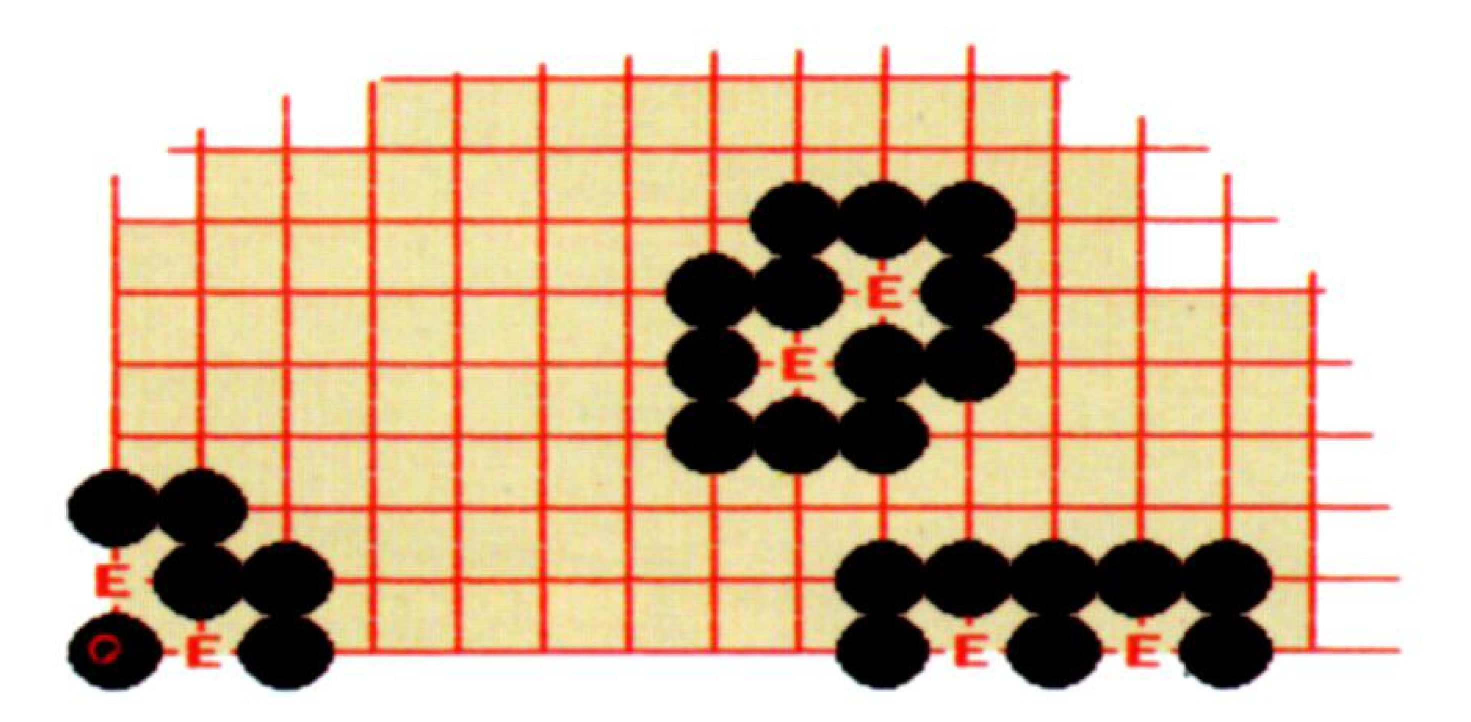
La forma más simple de hacer esto sería limitarse a efectuar cualquier movimiento al azar. Esta rutina sería algo así como:

```
5000 DEF PROCmovimient_azar
5010 LOCAL L%
5020 :
5030 FOR L%=17 TO 255
5040 IF FNlegalidad (L%,negras%)=0 THEN
     posición%=L%
5050 NEXT
5060 ENDPROC
```

Esta rutina devolverá un movimiento legal, en caso de que exista alguno. La rutina se podría mejorar comenzando el bucle en una posición aleatoria del tablero y, posiblemente, asignándole a la ficha colocada (L%) un marcador basado en la cantidad de licencias disponibles; éste se calculará automáticamente y FNlegalidad lo devolverá a la variable clib%. No obstante, aun con estas mejoras, el ordenador no va a efectuar movimientos muy sensatos, ¡cómo no sea por pura casualidad!

Lo que realmente se necesita es una rutina que efectúe movimientos al azar, pero sólo a posiciones razonablemente sensatas. Una forma de alcanzar este objetivo consiste en "evaluar" el tablero. Ésta es una técnica que se utiliza con frecuencia en juegos como "Otelo", en los que ciertas posiciones del tablero son más ventajosas que otras.

Aunque es difícil decidir las ventajas y desventajas de posiciones específicas, se considera que ciertas zonas del tablero son mejores que otras. Por ejemplo, consideremos la cantidad de piezas necesarias para rodear por completo dos licencias separadas. Recordará que se las denomina *ojos* (E) y que cualquier grupo con dos o más ojos se halla a salvo de captura. El número de fichas que se necesitan para formar un grupo seguro de esta manera es variable, según el grupo se halle en la esquina, a un lado o en el centro del tablero. Para formar un grupo seguro en la esquina sólo se requieren seis fichas y esto, obviamente, es mejor que las doce fichas que se precisan en medio del tablero.




Los primeros movimientos más probables serán en o cerca de las posiciones de inferioridad de la esquina. Desde aquí, en el curso habitual del juego, los jugadores se extienden a lo largo de los lados del tablero (en las líneas tercera o cuarta) en un intento por ganar territorio. Luego, de ser atacado, usted tiene la ventaja de poder utilizar su ficha de esquina de cuarto rango como base desde donde conseguir dos ojos en la esquina.

Habiendo explicado estas tácticas básicas, las estimaciones del tablero que se ofrecen en el diagrama tendrán algún sentido.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 4 | 3 | 2 | 1 | 0 |
| 2 | 3 | 5 | 5 | 4 | 3 | 2 | 1 |
| 2 | 4 | 6 | 5 | 5 | 4 | 3 | 2 |
| 2 | 4 | 6 | 6 | 6 | 5 | 4 | 3 |
| 2 | 5 | 7 | 7 | 6 | 5 | 5 | 4 |
| 1 | 5 | 7 | 7 | 6 | 6 | 5 | 5 |
| 1 | 4 | 5 | 5 | 4 | 4 | 3 | 3 |
| 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |

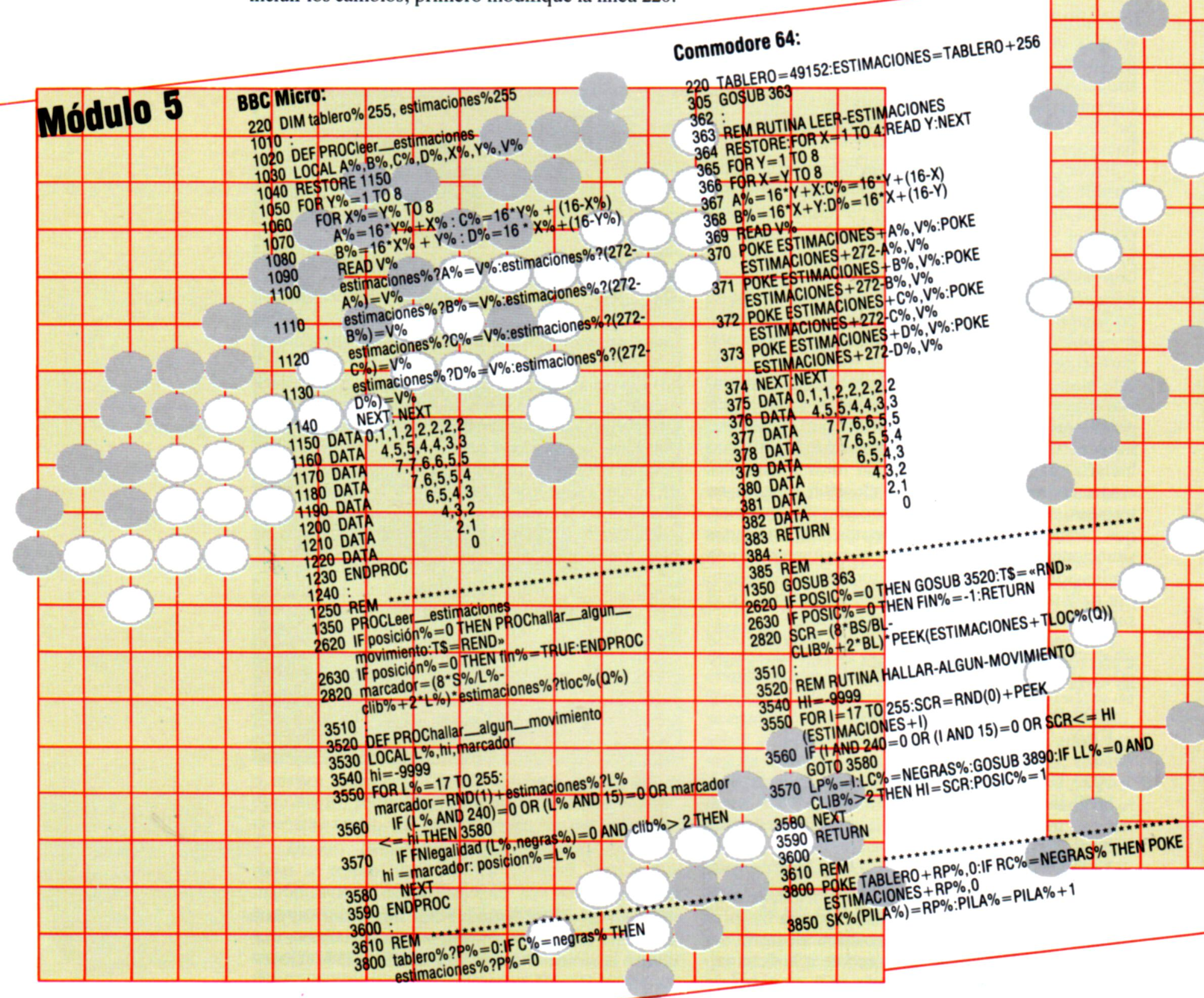
Éstas son simétricas para las cuatro esquinas del tablero, estableciéndose mediante la rutina entre las líneas 1020-1230. (Observe que en la versión para el C64, la rutina se consigna fuera de secuencia, entre las líneas 363 y 383, debido a la falta de una instrucción RESTORE numero-linea para restaurar el puntero de datos.)

Los números asignados a las diversas posiciones se han escogido sobre la base de las tácticas que acabamos de describir, pero de ningún modo son los mejores. Quizá quiera probar estimaciones diferentes para comprobar su efecto. Una forma de

"regular" estas estimaciones consiste en hacer que el ordenador juegue consigo mismo, con cada bando usando una tabla de estimaciones diferente. Esto se puede hacer de la misma forma en que establecíamos al ordenador como árbitro en un juego para dos jugadores. Simplemente modifique las líneas 60 y 80 para que recen:

```
60 movimiento%=movimiento%+1:negras%=2:
   blancas%=1:PROCmovimiento__negras
80 movimiento%=movimiento%+1:negras%=1:
   blancas%=2:PROCmovimiento__negras
```

(Ver el recuadro *Complementos al* BASIC para los equivalentes para Amstrad, C64 y Spectrum.) Para incluir los cambios, primero modifique la línea 220:



## Módulo 5

**BBC Micro:**

```
220  DIM tablero% 255, estimaciones%255
1010 :
1020 DEF PROCleer__estimaciones
1030 LOCAL A%,B%,C%,D%,X%,Y%,V%
1040 RESTORE 1150
1050 FOR Y%=1 TO 8
1060      FOR X%=Y% TO 8
1070         A%=16*Y%+X% : C%=16*Y% + (16-X%)
1080         B%=16*X% + Y% : D%=16 * X%+(16-Y%)
1090         READ V%
1100         estimaciones%?A%=V%:estimaciones%?(272-
             A%)=V%
1110         estimaciones%?B%=V%:estimaciones%?(272-
             B%)=V%
1120         estimaciones%?C%=V%:estimaciones%?(272-
             C%)=V%
1130         estimaciones%?D%=V%:estimaciones%?(272-
             D%)=V%
             NEXT: NEXT
1140
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA   4,5,5,4,4,3,3
1170 DATA     7,7,6,6,5,5
1180 DATA       7,6,5,5,4
1190 DATA         6,5,4,3
1200 DATA           4,3,2
1210 DATA             2,1
1220 DATA               0
1230 ENDPROC
1240 :
1250 REM ****************************
1350 PROCLeer__estimaciones
2620 IF posición%=0 THEN PROChallar__algun__
     movimiento:T$=REND»
2630 IF posición%=0 THEN fin%=TRUE:ENDPROC
2820 marcador=(8*S%/L%-
     clib%+2*L%)*estimaciones%?tloc%(Q%)
3510 :
3520 DEF PROChallar__algun__movimiento
3530 LOCAL L%,hi,marcador
3540 hi=-9999
3550 FOR L%=17 TO 255:
     marcador=RND(1)+estimaciones%?L%
3560    IF (L% AND 240)=0 OR (L% AND 15)=0 OR marcador
     <=hi THEN 3580
3570    IF FNlegalidad (L%,negras%)=0 AND clib%> 2 THEN
     hi=marcador: posicion%=L%
3580    NEXT
3590 ENDPROC
3600 :
3610 REM ****************************
3800 tablero%?P%=0:IF C%=negras% THEN
     estimaciones%?P%=0
```

**Commodore 64:**

```
220  TABLERO=49152:ESTIMACIONES=TABLERO+256
305  GOSUB 363
362  :
363  REM RUTINA LEER-ESTIMACIONES
364  RESTORE:FOR X=1 TO 4:READ Y:NEXT
365  FOR Y=1 TO 8
366  FOR X=Y TO 8
367  A%=16*Y+X:C%=16*Y+(16-X)
368  B%=16*X+Y:D%=16*X+(16-Y)
369  READ V%
370  POKE ESTIMACIONES+A%,V%:POKE
     ESTIMACIONES+272-A%,V%
371  POKE ESTIMACIONES+B%,V%:POKE
     ESTIMACIONES+272-B%,V%
372  POKE ESTIMACIONES+C%,V%:POKE
     ESTIMACIONES+272-C%,V%
373  POKE ESTIMACIONES+D%,V%:POKE
     ESTIMACIONES+272-D%,V%
374  NEXT:NEXT
375  DATA 0,1,1,2,2,2,2,2
376  DATA   4,5,5,4,4,3,3
377  DATA     7,7,6,6,5,5
378  DATA       7,6,5,5,4
379  DATA         6,5,4,3
380  DATA           4,3,2
381  DATA             2,1
382  DATA               0
383  RETURN
384  :
385  REM ****************************
1350 GOSUB 363
2620 IF POSIC%=0 THEN GOSUB 3520:T$=«RND»
2630 IF POSIC%=0 THEN FIN%=-1:RETURN
2820 SCR=(8*BS/BL-
     CLIB%+2*BL)*PEEK(ESTIMACIONES+TLOC%(Q))
3510 :
3520 REM RUTINA HALLAR-ALGUN-MOVIMIENTO
3540 HI=-9999
3550 FOR I=17 TO 255:SCR=RND(0)+PEEK
     (ESTIMACIONES+I)
3560 IF (I AND 240=0 OR (I AND 15)=0 OR SCR<=HI
     GOTO 3580
3570 LP%=I:LC%=NEGRAS%:GOSUB 3890:IF LL%=0 AND
     CLIB%>2 THEN HI=SCR:POSIC%=1
3580 NEXT
3590 RETURN
3600 :
3610 REM ****************************
3800 POKE TABLERO+RP%,0:IF RC%=NEGRAS% THEN POKE
     ESTIMACIONES+RP%,0
3850 SK%(PILA%)=RP%:PILA%=PILA%+1
```

```
220 DIM tablero%255, estimación1%255,
    estimación2%255
```

La rutina PROCleer__estimaciones se puede, entonces, modificar para colocar un conjunto de estimaciones en la matriz de bytes estimaciones1%, y el otro en estimaciones2%. Ahora, añadiendo estimaciones%=estimaciones1% en la línea 60, y estimaciones%=estimaciones2% en la línea 80, el ordenador utilizará diferentes estimaciones del tablero para cada uno de los jugadores.

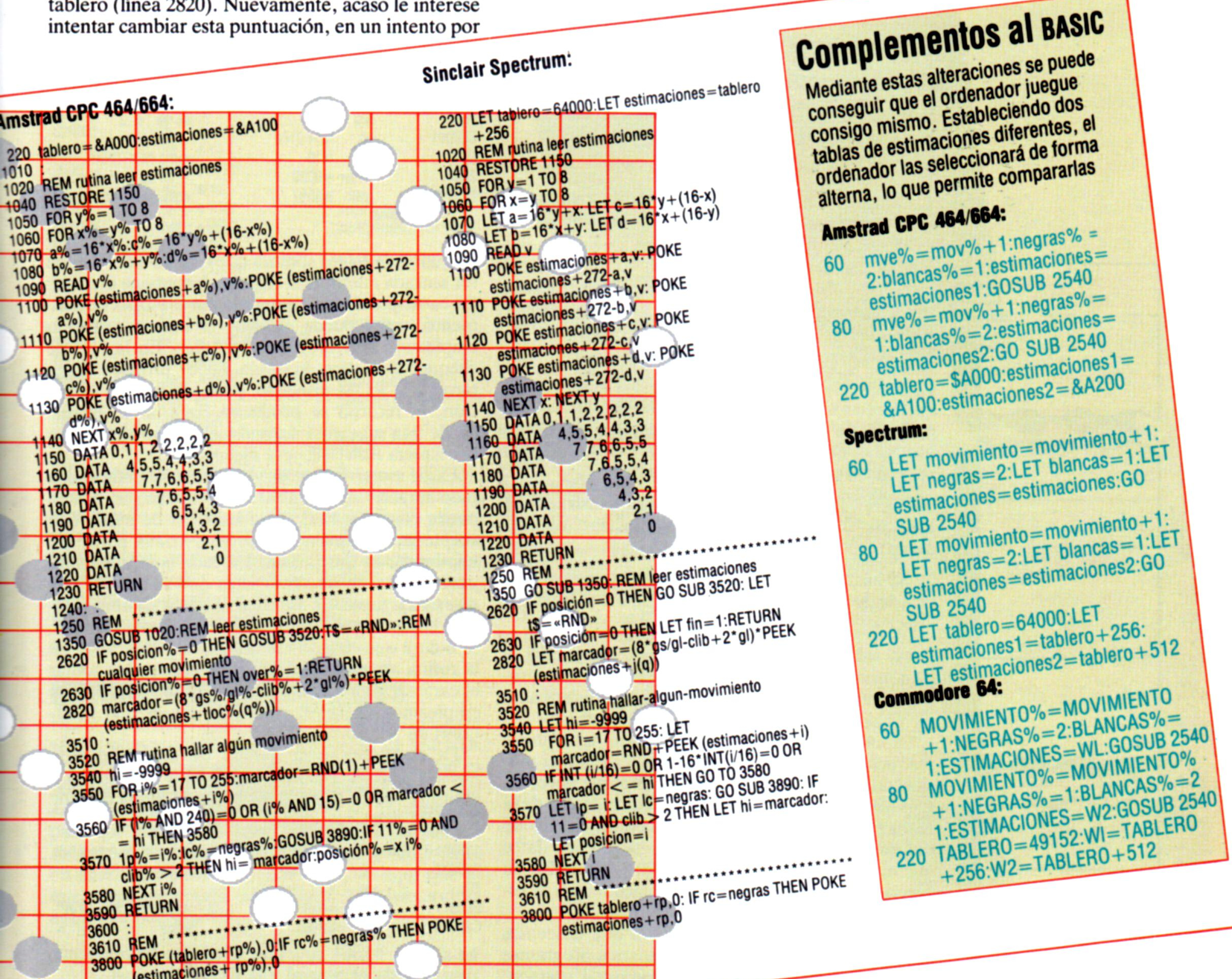
Habiendo calculado el tablero estimado, podemos añadir la rutina «general» PROC hallar__algun__movimiento. Es similar a la rutina de movimiento al azar, pero usa las estimaciones del tablero para hallar un movimiento «razonable». También se comprueba la variable clib%, para asegurar que el ordenador no juegue a una posición que deje a uno de sus grupos con menos de tres licencias. La línea 2620, con la observación "RND", llama a esta rutina. La indicación alude a la rutina que se utilizó para hallar el último movimiento del ordenador, de la misma forma que el comentario GP de la rutina de evaluación de grupos anterior. Si esta nueva ru-

tina no puede hallar ningún movimiento posible, entonces la línea 2630 establece la variable de fin del juego y sale del programa.

Ya que nos hemos molestado en establecer estimaciones del tablero, también podemos utilizarlas para mejorar otras rutinas de evaluación. En la rutina de evaluación de grupos, esto se puede hacer multiplicando la puntuación por la estimación del tablero (línea 2820). Nuevamente, acaso le interese intentar cambiar esta puntuación, en un intento por conseguir mejorar el rendimiento del ordenador.

Un problema evidente fue la tendencia del ordenador a jugar fichas en zonas de fichas previamente capturadas. Sobre la base de nuestras evaluaciones, obviamente las primeras fichas que se jueguen se colocarán en lo que el programa considera las mejores posiciones del tablero. Si usted captura algunas de estas fichas, entonces estas posiciones previamente "buenas" quedan vacantes. El ordenador, al percatarse de esto, inmediatamente comienza a jugar fichas otra vez en estas zonas, que como es evidente ya están rodeadas por sus fichas, lo cual no es una buena idea. En consecuencia, se ha añadido la línea 3800 para ofrecer una forma sencilla de estimación "dinámica" del tablero. Esto significa tan sólo que las estimaciones del tablero van cambiando a medida que avanza el juego. Por ejemplo, en el ajedrez usted evalúa el tablero para el rey. Durante los primeros movimientos del juego desearía estimaciones altas en la esquina para favorecer que el rey ocupe una posición segura. Sin embargo, al llegar al final de la partida, probablemente querrá cambiar las estimaciones para tentar al rey a una posición central más dominante.

En nuestro programa de *go*, la línea 3800 forma parte de la rutina PROCsuprimir, que suprime las fichas capturadas del tablero. Si estas fichas son negras, esta línea también cambiará la estimación del tablero para esa posición a cero. A partir de entonces, a pesar de que las negras aún pueden jugar en esta posición, esto es muchísimo menos probable.



**Amstrad CPC 464/664:**

```
220  tablero=&A000:estimaciones=&A100
1010 :
1020 REM rutina leer estimaciones
1040 RESTORE 1150
1050 FOR y%=1 TO 8
1060 FOR x%=y% TO 8
1070 a%=16*x%:c%=16*y%+(16-x%)
1080 b%=16*x%+y%:d%=16*x%+(16-x%)
1090 READ v%
1100 POKE (estimaciones+a%),v%:POKE (estimaciones+272-
       a%),v%
1110 POKE (estimaciones+b%),v%:POKE (estimaciones+272-
       b%),v%
1120 POKE (estimaciones+c%),v%:POKE (estimaciones+272-
       c%),v%
1130 POKE (estimaciones+d%),v%:POKE (estimaciones+272-
       d%),v%
1140 NEXT x%,y%
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA   4,5,5,4,4,3,3
1170 DATA     7,7,6,6,5,5
1180 DATA       7,6,5,5,4
1190 DATA         6,5,4,3
1200 DATA           4,3,2
1210 DATA             2,1
1220 DATA               0
1230 RETURN
1240: :
1250 REM ****************************************
1350 GOSUB 1020:REM leer estimaciones
2620 IF posicion%=0 THEN GOSUB 3520:T$=«RND»:REM
       cualquier movimiento
2630 IF posicion%=0 THEN over%=1:RETURN
2820 marcador=(8*gs%/gl%-clib%+2*gl%)*PEEK
       (estimaciones+tloc%(q%))
3510 :
3520 REM rutina hallar algún movimiento
3540 hi=-9999
3550 FOR i%=17 TO 255:marcador=RND(1)+PEEK
       (estimaciones+i%)
3560 IF (i% AND 240)=0 OR (i% AND 15)=0 OR marcador <
       = hi THEN 3580
3570 1p%=i%:lc%=negras%:GOSUB 3890:IF 11%=0 AND
       clib% > 2 THEN hi= marcador:posición%=x i%
3580 NEXT i%
3590 RETURN
3600 :
3610 REM ****************************************
3800 POKE (tablero+rp%),0:IF rc%=negras% THEN POKE
       (estimaciones+ rp%),0
```

**Sinclair Spectrum:**

```
220  LET tablero=64000:LET estimaciones=tablero
       +256
1020 REM rutina leer estimaciones
1040 RESTORE 1150
1050 FOR y=1 TO 8
1060 FOR x=y TO 8
1070 LET a=16*y+x: LET c=16*y+(16-x)
1080 LET b=16*x+y: LET d=16*x+(16-y)
1090 READ v
1100 POKE estimaciones+a,v: POKE
       estimaciones+272-a,v
1110 POKE estimaciones+b,v: POKE
       estimaciones+272-b,v
1120 POKE estimaciones+c,v: POKE
       estimaciones+272-c,v
1130 POKE estimaciones+d,v: POKE
       estimaciones+272-d,v
1140 NEXT x: NEXT y
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA   4,5,5,4,4,3,3
1170 DATA     7,7,6,6,5,5
1180 DATA       7,6,5,5,4
1190 DATA         6,5,4,3
1200 DATA           4,3,2
1210 DATA             2,1
1220 DATA               0
1230 RETURN
1250 REM ****************************************
1350 GO SUB 1350: REM leer estimaciones
2620 IF posición=0 THEN GO SUB 3520: LET
       t$=«RND»
2630 IF posición=0 THEN LET fin=1:RETURN
2820 LET marcador=(8*gs/gl-clib+2*gl)*PEEK
       (estimaciones+j(q))
3510 :
3520 REM rutina hallar-algun-movimiento
3540 LET hi=-9999
3550   FOR i=17 TO 255: LET
       marcador=RND+PEEK (estimaciones+i)
3560 IF INT (i/16)=0 OR 1-16*INT(i/16)=0 OR
       marcador <= hi THEN GO TO 3580
3570 LET lp= i: LET lc=negras: GO SUB 3890: IF
       11=0 AND clib > 2 THEN LET hi=marcador:
       LET posicion=i
3580 NEXT i
3590 RETURN
3610 REM ****************************************
3800 POKE tablero+rp,0: IF rc=negras THEN POKE
       estimaciones+rp,0
```

## Complementos al BASIC

**Mediante estas alteraciones se puede conseguir que el ordenador juegue consigo mismo. Estableciendo dos tablas de estimaciones diferentes, el ordenador las seleccionará de forma alterna, lo que permite compararlas**

**Amstrad CPC 464/664:**

```
60  mve%=mov%+1:negras% =
      2:blancas%=1:estimaciones=
      estimaciones1:GOSUB 2540
80  mve%=mov%+1:negras%=
      1:blancas%=2:estimaciones=
      estimaciones2:GO SUB 2540
220 tablero=$A000:estimaciones1=
      &A100:estimaciones2=&A200
```

**Spectrum:**

```
60  LET movimiento=movimiento+1:
      LET negras=2:LET blancas=1:LET
      estimaciones=estimaciones:GO
      SUB 2540
80  LET movimiento=movimiento+1:
      LET negras=2:LET blancas=1:LET
      estimaciones=estimaciones2:GO
      SUB 2540
220 LET tablero=64000:LET
      estimaciones1=tablero+256:
      LET estimaciones2=tablero+512
```

**Commodore 64:**

```
60  MOVIMIENTO%=MOVIMIENTO
      +1:NEGRAS%=2:BLANCAS%=
      1:ESTIMACIONES=WL:GOSUB 2540
80  MOVIMIENTO%=MOVIMIENTO%
      +1:NEGRAS%=1:BLANCAS%=2
      1:ESTIMACIONES=W2:GOSUB 2540
220 TABLERO=49152:WI=TABLERO
      +256:W2=TABLERO+512
```

# Por aquí, por favor

## Veamos cómo se pueden añadir nuevas instrucciones al BASIC mediante la ROM de la Interface 1

Toda instrucción nueva que usted desee añadir al BASIC del Spectrum tendrá primero que romper los chequeos sintácticos y los chequeos en fase de ejecución que llevan a cabo tanto la ROM principal como la ROM de la Interface. En este caso, el control se pasa a la rutina cuya dirección está contenida en la variable de sistema VECTOR en 23735 y 23736. Hay dos formas de hacer esto.

1. Se modifica una palabra clave para que rompa los chequeos. Esto se logra pasando un número incorrecto de parámetros, sacándolo de su sintaxis normal o añadiendo un carácter (llamado *breaker*) para generar un error. Por ejemplo:

BORDER * (se añade un *breaker* tras la instrucción)
LINE 10,10,1,19 (demasiados parámetros especificados)

2. Se añade una instrucción totalmente nueva precedida de una palabra orden con un *breaker* (como *BEEP o * PRINT).

Estas instrucciones tendrán acceso, claro está, a cualquier variable del BASIC que usted desee. Nosotros nos serviremos del segundo método para añadir nuevas instrucciones.

Supongamos que queremos añadir una nueva instrucción llamada *B, para obtener una nota musical durante un segundo. Lo primero que necesitamos hacer es decidir adónde irá a parar dentro de la memoria el nuevo código. No puede ir dentro de una sentencia REM, dado que la inserción de datos que realizan las variables es imprevisible. En su lugar emplearemos CLEAR nn para ganar algo de espacio.

Para cualquier nueva instrucción que deseemos añadir necesitamos estructurar nuestros programas para incluir los pasos que ilustramos al margen. Cuando se ha entrado la rutina a la que apunta VECTOR después de que el intérprete del BASIC ha encontrado un error, precisamos conservar el carácter en el texto del programa en BASIC que ha causado la anomalía para cerciorarnos si de hecho se trata de una de nuestras instrucciones adicionales. A este carácter apunta la variable de sistema CHADD (situada en 23755 y 23756).

Para recuperar el carácter al que apunta CHADD, necesitamos dos rutinas de la ROM principal: GETCHAR y NEXTCHAR. Las operaciones de estas rutinas vienen descritas con detalle en el recuadro *Direcciones útiles* (página contigua), y mientras se está en el entorno de la ROM sombra pueden ser llamadas, tanto la una como la otra, mediante RST#10 seguido de la dirección (bien #00188, bien #0020) de la rutina requerida.

Al entrar a la rutina apuntada por VECTOR, CHADD contendrá la dirección del carácter que ha generado el error. Por ello, para ejecutar nuestra sencilla instrucción *B necesitamos ensamblar este listado:

```
D7       vector:rst  #10
1800            defw #0018        ; dirección de GETCHAR
FE2A            cp   "*"          ;es un "*"?
C2F001          jp   nz,#01F0     ;si no, error
D7              rst  #10
2000            defw #0020        ;dirección de NEXTCHAR
FE42            cp   "B"          ;es un "B"?
C2F001          jp   nz,#01F0     ;si no, error
D7              rst  #10          ;toma NEXTCHAR, sería
2000            defw #0020        ;...fin de sentencia
CDB705          call #05B7        ;comprueba fin de sent.
         runtime:
```

La rutina de ROM sombra que está en #05B7 debe ser entrada mientras el registro A contiene el carácter correspondiente, apuntado por CHADD. Naturalmente, usted puede alcanzar el mismo resultado empleando NEXTCHAR, como antes. En la fase de chequeo sintáctico, la rutina #05B7 provoca un retorno a la ROM principal, pero, en fase de ejecución, el retorno se producirá como si se hiciese desde una subrutina normal y el código situado en la etiqueta RUNTIME será ejecutado.

Es de notar la sencillez con que se comprueba cada carácter del nombre de la instrucción. Usted puede construir fácilmente la rutina de modo que acepte minúsculas y mayúsculas como parte del nombre de la instrucción. En cada comprobación, para el caso de no ser la letra que se espera, hay que salir a través de #01F0 a la ROM sombra (versión 1) para significar un error de sintaxis. También se notará que cuando entramos por primera vez en la rutina apuntada por VECTOR, precisamos tomar el carácter actual apuntado por CHADD, por lo que empleamos GETCHAR en #0018 mejor que NEXTCHAR en #0020.

La codificación en tiempo de ejecución para nuestra sencilla rutina es inmediata y se limita a emitir el sonido musical mediante la rutina *beep* de la ROM principal, como se muestra ahora:

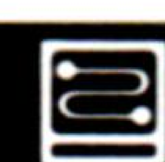
```
110501   runtim:ld   de,#0105     ;duración de 1 segundo
210007          ld   hl,#0700     ;altura
D7              rst  #10
B503            defw #03B5        ;ejecuta la rutina BEEP
C3C105          jp   #05C1        ;salida
```

El salto a #05C1 concluye el asunto limpiamente, devolviendo el control a la ROM principal.

Naturalmente, usted puede añadir una instrucción nueva al BASIC que se sirva de otras rutinas de la ROM principal ya estudiadas a lo largo de esta serie de análisis (además de las rutinas ROM para gráficos que veremos en el próximo y último capítulo), llamándolas por medio de RST#10. Como ejemplo servirá el siguiente listado para la instrucción *B, que se sirve de las rutinas en #1C82 y #1E94 (ambas se detallan en el recuadro *Direcciones útiles*) para evaluar un único parámetro numérico proporcionado por el usuario. El formato de las instrucciones *B n, donde n es cualquier número entre 0 y 255. El parámetro determina la longitud de BEEP en segundos, por lo que *B 255 genera un sonido de 255 segundos de duración.




**Un «caza» de la Interface**
El Spectrum responde a un error con una RST #08. Tan pronto como el hardware de la Interface 1 detecta que el contador de programas del Z80 contiene esta dirección, pagina la ROM de la Interface 1 y se realiza un salto a la rutina cuya dirección está contenida en VECTOR. Esto nos permite añadir nuevas instrucciones al Spectrum, empleando una estructura de programa como el que aquí se ilustra



```
                org   60000
         VECTOR:equ   #5CB7          ;dirección de VECTOR
CF          init:rst  #8
31              defb  49             ;establece variables IF1
2169EA          ld    hl,newcom      ;añade nueva instrucción
22B75C          ld    (VECTOR), hl   ;lo toma en VECTOR
C9              ret                  ;fin de inicialización
        ;
D7       newcom:rst   #10            ;VECTOR salta aquí
1800            defw  #0018          ;GETCHAR
FE2A            cp    "*"
C2F001          jp    nz,#01F0
D7              rst   #10
2000            defw  #0020          ;NEXTCHAR
FE42            cp    "B"
C2F001          jp    nz,#01F0
D7              rst   #10
2000            defw  #0020
D7              rst   #10
821C            defw  #1C82          ;evalúa parámetro
CDB705          call  #05B7          ;comprueba final sent.
D7       runtim:rst   #10
941E            defw  #1E94          ;toma parám. en a
47              ld    b,a
C5       loop:  push  bc             ;guarda contador
110501          ld    de,#0105       ;duración
210007          ld    hl,#0700       ;altura
D7              rst   #10
B503            defw  #03B5          ;llama rutina BEEP
C1              pop   bc             ;restaura contador
10F3            djnz  loop
C3C105          jp    #05C1          ;salida - final
```

Obsérvese que a causa de que la rutina en #1C82 sólo acepta un valor de ocho bits (ya que este valor es almacenado en el registro A), la entrada de valores fuera del intervalo de 0 a 255 generará error. Otro punto a tener en cuenta es que, durante la generación de notas musicales, el contador FRAMES no se incrementará y no será explorado el teclado, y, por tanto, no hay que entrar el 255 como parámetro a menos que esté dispuesto a esperar un poco...

## Parámetros adicionales

Hasta aquí nos hemos ceñido en el método para añadir nuevas instrucciones por medio de entradas no habituales que generen error. Sin duda que es posible también emplear las palabras clave del BASIC Sinclair, aunque habrán de ser modificadas para que generen el error perseguido. La manera más sencilla de conseguirlo consiste generalmente en añadir un parámetro adicional, de aquí que lo que sigue provocará un salto a cualquier rutina apuntada por VECTOR:

```
CIRCLE x,y,z,n
LINE x
BORDER x,y,z
```

Si usted opta por este camino, CHADD apuntará al indicativo (*token*) de la palabra clave irregular en el momento de entrar la rutina apuntada por VECTOR. Esto puede comprobarse con el código del *token* adecuado, que puede encontrarse en el manual del Spectrum.

A pesar de la complejidad aparente del procedimiento, la adición de nuevas palabras clave al BASIC del Spectrum es de hecho bastante fácil. Lo que importa es recordar qué ROM está paginada en cada momento. Si ha de abandonar la ROM sombra, podrá emplear el código de enganche 50, más la dirección adecuada, para llamar la rutina de la ROM sombra.

## Identificación

Si trata de acceder directamente a las rutinas de la ROM sombra, le interesa comprobar antes cuál es la versión de la ROM de la Interface 1 que está usando. Por regla general, las IF1 con números de serie superiores a 87315 llevan la versión 2 de la ROM, pero una manera más segura de comprobarlo es entrar esto:

```
CLOSE #0; PRINT PEEK 23729
```

Que dará 0 tratándose de la versión 1 y 80 en la versión 2. Por suerte, las llamadas de la ROM de la IF1 que hemos utilizado para ampliar el BASIC se encuentran en la misma dirección tanto en una como en otra versión. Aun así aparecerán problemas cuando emplee otras rutinas. En caso de duda, y suponiendo que posea un buen desensamblador, puede que prefiera comprobar las rutinas que usted llama empleando el programa *Cargador de la ROM sombra* que ya hemos publicado. Es de notar la adición de dos nuevos códigos de enganche en la ROM de la versión 2. El código de enganche #33 recupera el descriptor de registros de un archivo en el microdrive, y #34 abre un canal «b» RS232. En este último caso, la dirección de base del canal se proporciona en DE

## Direcciones útiles

**En la ROM sombra:**

| | |
|---|---|
| #01F0 | Abandona la ROM sombra a través de una rutina de error de la ROM principal |
| #05B7 | Efectúa un retorno en tiempo de ejecución si el car. contenido en el reg. A es #0D (ENTER) o un punto y coma. De lo contrario genera un error |
| #05C1 | Retorna al intérprete principal cuando ha sido aceptada una nueva instr. |

**En la ROM principal:**

| | |
|---|---|
| #0018 | GETCHAR: Toma CHADD (ver abajo) en HL; toma el car. al que apunta CHADD en A; mira si es un car. imprimible y hace un retorno si lo es. De lo contrario pasa a NEXTCHAR |
| #0020 | NEXTCHAR: Incrementa CHADD y lo coloca en HL. Toma el carácter apuntado por CHADD y comprueba si es imprimible. Si lo es, entonces NEXTCHAR retorna con el carácter an A. Si no lo es, se repite el proceso |
| #1C82 | Evalúa o chequea una expresión numérica, cuyo primer elemento debe ser el apuntado por CHADD. A la salida, CHADD apunta al primer carácter que sigue a la expresión, y el resultado se coloca en la parte superior de la pila calculadora de la ROM principal |
| #1E94 | Toma el valor de la parte superior de la pila de cálculo de la ROM principal y lo almacena en A si está dentro del intervalo de 0 a 255. De lo contrario, genera un error |

La variable de sistema CHADD apunta al carácter que se está interpretando, y se sitúa en las direcciones 23755 y 23756

# En pos del triunfo

## Los juegos de estrategia exigen conjugar numerosos factores y adoptar juiciosas decisiones. «Football manager» constituye un buen ejemplo

Lanzado al mercado a principios de los ochenta, el programa *Football manager* (Entrenador de fútbol) ha generado constantes ventas a largo plazo. En un mercado en el cual la mayoría de los paquetes permanecen en las estanterías de las tiendas sólo durante un período limitado, independientemente del éxito que obtengan al editarse, este juego destaca entre sus competidores. Y, además, no tiene nada particularmente innovador: el concepto del juego es muy simple. Usted dirige un equipo de fútbol y debe conducirlo al éxito a lo largo de la Liga.




Liz Heaney

**Impaciente y nerviosa espera**
Esta pantalla, tomada de la versión de *Football manager* para el Commodore 64, muestra la "secuencia de acciones" del juego. En este punto, el jugador no tiene control alguno sobre los incidentes que se producen en el campo de juego, sino que se limita a contemplar "las jugadas más interesantes" del encuentro. Ésta quizá sea la secuencia del juego más llena de nerviosismo y emoción, puesto que usted permanece a la espera de ver si sus evaluaciones han sido correctas

El juego comienza, como es natural, al iniciarse una nueva temporada, con el equipo que usted dirige en la Cuarta División. Luego se le presenta un menú de opciones tales como comprobar su posición en la tabla de la Liga, ver detalles de la lista de encuentros y considerar las características de los jugadores de que dispone para integrar el equipo.

Los jugadores se dividen en tres categorías: defensas, centrocampistas y delanteros. Como es obvio, un entrenador sensato formará un equipo equilibrado para cada encuentro. Antes de un partido se le ofrece a usted la opción de introducir modificaciones en la alineación, con el fin de sacar al campo el equipo más adecuado. No obstante, en las primeras etapas del juego sólo dispone de 12 jugadores, por lo cual no habrá mucho donde escoger. Sin embargo, a medida que el juego avanza, aparecen jugadores adicionales provenientes del mercado de traspasos y puede "pujar" por ellos.

Un jugador posee tres características básicas, que se deben tener en cuenta al pujar. En primer lugar, cada uno de ellos posee un "factor de destreza", medido en la escala entre uno y cinco. Segundo, cada uno tiene un "valor" que, aunque relacionado con el factor de destreza (5 000 por cada punto de destreza), puede variar en el mercado de traspasos. Por último, está la "energía" del deportista. Ésta se halla comprendida entre 1 y 20 y disminuye cada vez que un jugador es alineado para un partido. Por lo tanto, no es aconsejable alinear a la totalidad de los mejores jugadores en todos los encuentros, sino que es mejor ir rotándolos de modo que puedan disfrutar de descansos periódicos para poder restablecer sus energías hasta el máximo. Por supuesto, esto es muy difícil con una plantilla de apenas doce jugadores, y, al enfrentarse a un factor de energía en disminución, aun el entrenador más cuidadoso se verá obligado a recurrir tarde o temprano al mercado de traspasos.

Al comienzo de cada "vuelta" se ofrecerá el traspaso de un jugador y se le invitará a pujar por él. Si se acepta su oferta, el deportista pasará a formar parte de la plantilla. Sin embargo, con frecuencia la oferta se rechazará, aun cuando ofrezca más del valor establecido. Entonces se le pedirá que haga otra oferta, sólo que esta vez el valor del traspaso del jugador aumentará debido a que usted se muestra muy interesado por él.

Tras terminar sus incursiones en el mercado de traspasos, acordando préstamos si fuera necesario, y después de examinar su posición actual en la Liga, se le dará la opción de jugar un partido. El ordenador visualiza las fortalezas relativas de su equipo y las del adversario, y usted tiene la opción de cambiar el equipo, sustituyendo los jugadores cansados y lesionados por los de refresco. Tras haber completado esto, el ordenador presentará las "jugadas más interesantes" del encuentro.

Las jugadas más interesantes consisten simplemente en las escaramuzas producidas en la zona de portería de ambos bandos; el tiempo que se dedique a una u otra portería dependerá de las fortalezas relativas de los centrocampistas, mientras que las fortalezas de los delanteros y de los defensores adversarios determinarán la consecución o no de los goles. Aunque los gráficos del partido le proporcionan al juego mucha emoción mientras se contemplan las incidencias del mismo, hay que decir que la visualización de gráficos es muy primitiva; la versión para el BBC, por ejemplo, se compone simplemente de caracteres gráficos en bloque.

Al terminar el encuentro se visualiza el marcador final, junto con los resultados de los otros partidos.

El juego requiere que el entrenador no sólo considere la mejor estrategia para el próximo encuentro, sino que también prevea los partidos venideros, tal como las fechas en las que hay de medirse con equipos que ocupen las primeras posiciones de la tabla y los encuentros coperos, en los que se requiere el mejor equipo posible. Por lo tanto, es necesario tomar decisiones acerca de concederles descanso a los mejores jugadores ahora (y arriesgarse a perder puntos) o hacerlos jugar (corriendo el riesgo de que se lesionen o pierdan energía, lo que incidiría negativamente en su rendimiento en los partidos importantes).

A pesar de que el juego no contiene secuencias "recreativas" propiamente dichas, es fácil comprender las razones por las que ha conservado su popularidad. A pesar de algunas limitaciones, es una simulación suficientemente realista como para lograr que el jugador crea realmente que está dirigiendo un equipo. Y cuando finalmente su equipo gane el campeonato de Primera División, usted se sentirá realmente satisfecho.

**Football manager:** Para el ZX81, Sinclair Spectrum, Commodore 64, Vic-20, BBC Micro, Electron, Amstrad y Dragon
**Editado por:** Addictive Games Ltd, 7A Richmond Hill, Bournemouth, Dorset BH2 6HE, Gran Bretaña
**Autor:** Kevin Toms
**Formato:** Cassette
**Palancas de mando:** No se necesitan

Editorial Delta, S.A.



10099

9 788485 822836